NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

LARGE SCALE CROSS-DRIVE CORRELATION OF DIGITAL MEDIA

by

Joseph Van Bruaene

March 2016

Thesis Co-Advisors: Michael McCarrin
Mark Gondree

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	03-31-2016	Master's Thesis 06-01-2015 to 03-31-2016

4. TITLE AND SUBTITLE

LARGE SCALE CROSS-DRIVE CORRELATION OF DIGITAL MEDIA

5. FUNDING NUMBERS

6. AUTHOR(S)

Joseph Van Bruaene

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: NPS.2012-0024-CR04-EP5-A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Traditional digital forensic practices have focused on individual hard disk analysis. As the digital universe continues to grow, and cyber crimes become more prevalent, the ability to make large scale cross-drive correlations among a large corpus of digital media becomes increasingly important. We propose a methodology that builds on bulk-analysis techniques to avoid operating system- and file-system specific parsing. In addition, we apply document similarity methods to forensic artifact correlation. By representing each disk image as a set of hash values corresponding to the 512-byte sectors on the disk, and calculating pair-wise similarity scores between hard disk images, we analyze a collection of disk images taken from various storage devices purchased from the secondary market. We conclude sector-based matching is sufficient to identify images in our dataset that share common DLLs, indicating similarity in their operating systems. We present a visualization of our results as an undirected graph with similarity scores represented as edge weights, and observe that disk images with common operating systems tend to align with graph clusters. Though no common set of sectors is present on all drives—even among the large fully-connected component in our graph—we find that grouping our dataset into subsets with the same operating system version does reveal sizable collections of common sectors, and achieved the best correlation between sector matches and high-level similarities in our dataset. Extending this technique to a larger dataset and continuing our investigation of the cause of sector-level matches could yield an automated method of profiling new disk images during the triage process. Moreover, this technique could be used to corroborate deductions regarding characteristics of information systems associated with target media.

14. SUBJECT TERMS

Digital Forensics, Similarity Detection, Automated Correlation, Digital Fingerprinting, Approximate Matching, Bulk Analysis

15. NUMBER OF PAGES 87

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

LARGE SCALE CROSS-DRIVE CORRELATION OF DIGITAL MEDIA

Joseph Van Bruaene
Lieutenant, United States Navy
MBA, Indiana University, 2008

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Approved by: Michael McCarrin
Thesis Co-Advisor

Mark Gondree
Thesis Co-Advisor

Peter Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Traditional digital forensic practices have focused on individual hard disk analysis. As the digital universe continues to grow, and cyber crimes become more prevalent, the ability to make large scale cross-drive correlations among a large corpus of digital media becomes increasingly important. We propose a methodology that builds on bulk-analysis techniques to avoid operating system- and file-system specific parsing. In addition, we apply document similarity methods to forensic artifact correlation. By representing each disk image as a set of hash values corresponding to the 512-byte sectors on the disk, and calculating pair-wise similarity scores between hard disk images, we analyze a collection of disk images taken from various storage devices purchased from the secondary market. We conclude sector-based matching is sufficient to identify images in our dataset that share common DLLs, indicating similarity in their operating systems. We present a visualization of our results as an undirected graph with similarity scores represented as edge weights, and observe that disk images with common operating systems tend to align with graph clusters. Though no common set of sectors is present on all drives—even among the large fully-connected component in our graph—we find that grouping our dataset into subsets with the same operating system version does reveal sizable collections of common sectors, and achieved the best correlation between sector matches and high-level similarities in our dataset. Extending this technique to a larger dataset and continuing our investigation of the cause of sector-level matches could yield an automated method of profiling new disk images during the triage process. Moreover, this technique could be used to corroborate deductions regarding characteristics of information systems associated with target media.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Acronyms and Abbreviations

BA	Bosnia and Herzegovina
BD	Bangladesh
CDF	cumulative distribution function
CN	China
CSIRT	Computer Security Incident Response Team
CSV	comma-separated values
CTPH	Context Triggered Piecewise Hashing
DLL	dynamic-link library
DOD	Department of Defense
HPC	High Performance Computing Center
HU	Hungary
IL	Israel
NPS	Naval Postgraduate School
PK	Pakistan
RDC	Real Data Corpus
TH	Thailand

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgments

This thesis would not be possible without the help and guidance of many. I would first like to thank my thesis advisors, Michael McCarrin and Mark Gondree. Michael, if it were not for your technical teaching both in the classroom and for my research, I would have never been able to get started. Your commitment to explaining countless topics related to digital forensics, often multiple times, enabled me to persevere and complete this project. Mark, your insights and attention to detail were instrumental in shaping the direction of this work. When we got down to the wire, you took the time to provide thoughtful feedback, often on the weekend, to ensure the completion of this thesis. I am eternally grateful to both of you for your dedication and passion for research.

I would like to thank my wife, Julie, for taking care of everything these last few months as I disappeared for countless nights and weekends to finish my research. I would like to thank my parents for always encouraging me to work hard and always do my best in everything I do. Thank you to my fellow students who suffered along with me during this process and helped me in so many ways.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

Applications of digital forensics range from investigation of both cyber and traditional crimes, through intelligence gathering efforts, to protection of intellectual property. In these capacities, digital forensic capabilities are increasingly relevant to a wide range of organizations, including law enforcement, intelligence agencies, the military, and both public and private institutions. As the need for these capabilities expands to more domains, practitioners struggle to keep pace with the proliferation of digital media and an increased number of digital artifacts requiring analysis.

Law enforcement and intelligence agencies have been using digital forensics to collect information and prosecute criminal activity for decades. Cyber crimes include the theft or destruction of intellectual property, child solicitation and pornography, misuse of information systems, and the distribution of malicious software. In addition to its direct use for the detection and prosecution of cyber crimes, digital evidence can also be used as supporting evidence to non-electronic crimes, where it can provide intent, whereabouts, and proof of relationships.

According to a cybersecurity market report issued by the research firm MarketsandMarkets, the global cyber security market (security solutions and services) is expected to grow from $63.7 billion in 2011 to $120.1 billion by 2017 [1]. In an effort to combat this new illicit activity, law enforcement agencies, financial institutions, and other private businesses are incorporating digital forensic experts into their organizations [2]. As courts prosecute more crimes related to computer systems, and establish legal precedents, digital forensics will be heavily affected by new court rulings. The impact for digital analysts will be that collection methods and scientific analysis be performed in a way that is legally admissible in the court of law [3]. Further, rising demand for analysis of digital evidence produces a corresponding need for improved methods of acquiring and understanding this evidence. As cyber crime and the amount of data increases, digital practitioners must be able to triage large amounts of data efficiently and accurately. In recognition of this development, the National Institute of Justice supports the development of automated tools and the collection of digital evidence through its Electronic Crime Program [4].

In addition to the applications related to law enforcement, digital forensics is also a key tool used by the military and public and private organizations to understand the security environment an organization is facing and investigate cyber security incidents. (In this context, an incident can be any act violating an explicit or implied security policy—e.g., unauthorized access, denial of service, destruction of hardware or software.) Here also, the proliferation of digital evidence in need of examination is contributing to the expanding demand for scalable, automatic forensic analysis techniques. The military environment often operates in real-time and cannot afford to take systems offline to investigate each breach of a system. Therefore, in the event of a computer security incident, digital forensics becomes a critical component of the collection and evaluation of a Computer Security Incident Response Team (CSIRT)—the entity responsible for reviewing the event. As the focal point for the incident, the CSIRT is the central location for all analysis and correlation of event activity. As the number of events grows within an organization and expertise is established within the CSIRT, digital forensic analysts need a way to correlate new incidents with past events to better understand the security environment. Depending on the scale of the incident, being able to sift through large amounts of data to determine relevant activity could be the determining factor in effective incident handling, as well as the efficient use of limited CSIRT manpower. Identifying trends and likely behavior in the triage process allows for more accurate handling and analysis by focusing on the key incident factors and filtering out irrelevant data. The interconnected environment of today's computer systems necessitates the ability to make correlations between disparate data sources for more efficient analysis [5].

Even as digital forensics is being applied to more domains, growth within these domains—and with respect to the digital universe in general—contributes to the rise of new technical challenges. According to EMC, a global leader in providing information technology as a service, the digital universe is doubling every two years. It is estimated that by 2020, the amount of data created will reach 44 zettabytes (44 trillion gigabytes) [6]. While most of this data is transient, and not stored, the accelerating production of data coupled with decreasing hard disk prices [7] vastly increases the amount of potential data a forensic practitioner must be able to collect and analyze. As more organizations monitor employees' computers for unauthorized activity, more law enforcement officers examine victim and suspect computers for incriminating evidence, more network analysts scrutinize communications to and from

client machines—in general, that is, as the amount of forensic data grows—we face an intensifying need for a methodology to incorporate multiple data sources to investigate similarities in processed digital media. With data growth rates approaching 40% per year for the next decade, digital forensics has the opportunity to provide actionable intelligence to otherwise dormant data with the development and use of tools and techniques able to correlate large amounts of data.

1.1 Motivation

As practitioners become inundated with greater amounts of data, they need a systematic way to retain the knowledge learned from past analysis. The ability of a digital forensic analyst to efficiently and accurately process relevant data becomes increasingly impaired as data becomes more ubiquitous and consumers move to multiple types of storage devices (thumb drives, external hard drives, cloud storage, mobile devices, tablets, etc.). Every time someone creates a new file system or operating system, we have to start over and maintaining a set of tools that can handle every different format is very difficult, requiring a mass community effort (an example is the Sleuth Kit). Even when this is coordinated effectively, we are still vulnerable to an adversary who recognizes that our resources can be overwhelmed by using unusual systems, or inventing new, custom systems for which tools do not yet exist.

In light of these challenges, it becomes increasingly important that tools and techniques be able to detect similarities and draw correlations between large amounts of disparate storage devices. Traditional digital forensic practices have focused on the analysis of digital media on a case-by-case basis. Typically, an analyst or team of analysts examine one hard drive image or small collection associated with a case, then move on to the next. With this process, even with detailed notes, it is difficult for an organization to recognize similarities and make correlations between cases, especially over long periods of time. The conventional approach to digital forensics limits the scope of analysis to individual devices and severely restricts the intelligence that can be derived from making large scale cross-drive correlations. An alternative approach consists of a matching and correlation process capable of being automated and run on large amounts of data.

We propose a methodology that builds on bulk-analysis techniques to avoid operating system,

and file-system specific parsing. In addition, we draw on general methods developed from document similarity work (which has been very effective for applications such as web search) and apply these to forensic artifact correlation. Because little research has been done in this area, we hope to offer preliminary groundwork for automatically identifying features and artifacts of forensic interest, which might eventually be used to profile new or unknown systems.

1.2 Contributions

This thesis combines sector-based matching techniques with similarity algorithms to correlate disk images with common sectors among a large corpus. We characterize these common sectors in terms of their frequency, and attempt to provide some insight into the high-level semantic similarities from which they derive. We conclude sector-based matching is sufficient to identify images sharing common dynamic-link library (DLL) files indicating the presence of shared operating system artifacts due to related operating system versions. Further, we present a visualization of our results as an undirected graph (with similarity scores as edge weights), which clusters a majority of the images in our dataset based on operating system and identifies small groups of uniquely matched images. We determine the disk images represented in the largest fully-connected component of our graph do not contain a single common set of sectors present on all images; however, further examination shows a sizable group of common sectors is shared by disk images tagged with the same operating system version.

Extending this technique to a larger dataset and continuing our investigation of the cause of sector-level matches could yield an automated method of profiling new disk images during the triage process, which would be especially useful if devices were acquired under circumstances that did not permit inspection of their running state. Moreover, this technique could be used to corroborate deductions regarding characteristics of information systems associated with target media. Such a capability could provide quick insight into community infrastructure or the characteristics of organizations from which drives were collected. Conversely, if common sectors corresponding to operating system structures were identified and removed, our method might prove useful in identifying other important similarities across large collections of drives, such as virus infections or common user data.

1.3 Outline of this Thesis

The remainder of this thesis is organized as follows. Chapter 2 provides a technical background. Chapter 3 describes previous work conducted. Chapter 4 describes the methodology for calculating our pair-wise similarity metrics and the experiments conducted to determine the underlying causes of the similarities. Chapter 5 describes the results. And finally, Chapter 6 discusses our conclusions and suggests future work.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 2:
Technical Background

To facilitate our explanation of our methodology of calculating pair-wise similarities between hard drive images and identifying high-level similarities among digital artifacts, we provide the technical background of three key areas:

1. Secondary storage basics,
2. Document similarity, and
3. Bytewise matching.

The first topic covers an introduction to some fundamental underlying concepts in digital media and its forensic representation as disk images, of which our dataset is comprised. Once this is established, we proceed to discuss some relevant techniques for measuring similarities between text documents, which we will generalize to arbitrary binary data. We also given an example of a method of visualizing similarity relationships using a graph. Lastly, we discuss several concepts relied on heavily in approximate matching techniques developed for digital forensics.

2.1 Secondary Storage Basics

All experiments we perform depend on a dataset comprised of hard drives purchased on the secondary market from all countries except the United States, and stored as a large collection of disk image files. To understand the storage devices we are evaluating, some familiarity with basic disk-level structures and disk images formats is helpful.

2.1.1 Disk Sector

A sector is the smallest unit of data that can be read or written to a disk. Each sector is a fixed amount of data that is read sequentially and is passed between the disk and its controller [8]. When the physical limits of magnetic disk technology were reached in 2009, manufacturers began migrating away from the traditional 512-byte sectors to 4096-byte sectors. As areal densities increased over time, the traditional smaller 512-byte sector began to consume less space on the hard drive surface, which impacted error correction and increased the risk of

media defects. The transition to 4096 bytes allowed hard drives to store the physical data more efficiently on the disk and provide more robust error detection [9]. This new disk format is known as the advanced format and was chosen to match the x86 page size and the default NTFS cluster size. The advanced format provides about 10% greater capacity but potentially wastes space for small files unless the filesystem can efficiently store them [10].

Sector size is important in digital forensics as analysts often choose equal-sized blocks for hashing based on the sector size of a given hard drive. However, for a large-scale similarity analysis of many drives, it is important that the sector size remain constant between processed media to preserve alignment. To compare fingerprints (see Section 2.3.1) for similarity across disparate sources, the input size must remain the same to ensure matching fingerprints correspond to identical objects. For our analysis, we have chosen to divide all drives into 512-byte sectors. In doing so, we will be able to process hard drive images regardless of the original disk's sector size by ensuring our granularity is small enough to include both sector sizes (a 4096-byte sector is eight 512-byte sectors).

2.1.2 Disk Image

A disk image is a sector-by-sector (see Section 2.1.1) software representation of a hard disk including the partition table, file allocation tables, and data partitions. A disk image is produced from the secondary storage device without reference to operating system level structures and enables a digital forensic practitioner to perform analysis on a copy of the original drive so that changes are not inadvertently applied during analysis thus preserving the integrity of the disk. With the original disk intact, it is then possible to create multiple disk images and reproduce forensic test results. A sector-by-sector copy may also preserve data not available to the operating system due to deliberate actions to hide data (e.g., hidden partitions) [11].

There are multiple types of disk images. Two examples include the raw and E01 format. The raw format is a generic standard produced by a sector-by-sector copy and is often identified by an img or raw extension. The E01 format is a proprietary and partially open specification format created by EnCase software to store digital evidence (identified by an E01 extension). The E01 format can store a single image in one or more segment files (e.g., E01, E02, E03) and is an industry standard for storing digital forensic images. Each file

consists of a standard 13-byte header, followed by a series of sections, where each section has a header that contains information such as case number, evidence number, unique description, examiner name, and notes [12]. The disk images used in our analysis were produced using the E01 format.

2.2 Document Similarity

Document similarity is an important class of problems with applications ranging from search to copyright enforcement to deduplication. Our discussion includes an introduction to types and tokens, covers the notions of resemblance and containment, and finishes with Jaccard similarity.

2.2.1 Types And Tokens

The distinction between a type and a token is the separation between a generic class of object (e.g., "bicycles") and a particular concrete instance (e.g., "my bicycle"). Broder uses a snippet of the Gertrude Stein poem *Sacred Emily* to illustrate the difference [13]:

A rose is a rose is a rose is a rose.

If asked to count the words in this sentence, one might answer that there are three words (a, rose, is) or ten words (a, rose, is, a, rose, is, a, rose, is, a, rose). Words in the first sense are considered types, abstract and unique, and words in the second sense tokens, concrete and particular [14]. For our analysis, we will differentiate our similarity metrics based on how we count each hash—as either a type or a token.

2.2.2 Resemblance and Containment

To compare two digital objects, we need to understand what those objects are. In document similarity, a document can be thought of as a collection of words, and each word as a part of a set that makes up the document. The set is often referred to as a bag, or multiset, in that words often repeat in documents. However, we can also represent each word only once and represent the document as a set. In both cases, we do not deal with the meaning of the text, but instead look for the presence of common words. Broder [13] discusses two ways to interpret similarity in this case:

1. Resemblance, showing two objects are roughly the same, and
2. Containment, showing one object is roughly contained in an other;

each of which can be reduced to a set intersection problem.

The resemblance of two sets, A and B, is defined as:

$$r(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.1}$$

This score is also referred to as the Jaccard similarity—see Section 2.2.3 [15]. The containment of two sets, A and B, is defined as:

$$c(A, B) = \frac{|A \cap B|}{|A|}. \tag{2.2}$$

For our analysis, the similarity score is calculated using the resemblance equation, where each disk image is treated as a set of hash values representing sectors, and the cardinality of the intersection of two sets is weighed against the cardinality of the union of both sets. Generally, we can calculate the similarity score, s, between two sets of hashed sectors, A and B, as:

$$s(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.3}$$

Depending on the cardinality of our two sets A and B, the resemblance and containment equations yield very similar results when the cardinality of A and B are roughly the same, and both approach 1 when A approaches B. The primary difference arises when handling objects where the cardinality of each set is of dramatically different sizes (probably because the objects themselves are different sizes). For our analysis, we are choosing a resemblance measure which will penalize comparisons between very small and very large drives due to the low cardinality of the intersection. However, the Jaccard similarity does provide some useful features for future work. For one, it is the complement of the Jaccard distance, which measures dissimilar sets, and may be efficiently estimated with the MinHash scheme [13].

2.2.3 Jaccard Similarity

Jaccard similarity is a statistic used for comparing sets by looking at the relative size of their intersection. Based on how we treat our hashes, as either types or tokens (see Section 2.2.1), we arrive at two different similarity metrics: a multiset Jaccard similarity and a set Jaccard similarity. The similarity score, s, can be represented as a function of two sets, A and B, where we divide the cardinality of the intersection of A and B with the cardinality of the union of A and B:

$$s(A, B) = \frac{|A \cap B|}{|A \cup B|}. \tag{2.4}$$

Because of a minor quirk in the tool we used to store and manipulate sets of hashes (see Section 4.3.2), our multiset Jaccard similarity differs slightly from the standard definition. Typically, in multisets, each element is assigned a multiplicity corresponding to the number of times that element occurs in the multiset. In traditional multiset intersection operations, each element in the intersection of the two sets has a multiplicity equal to the minimum multiplicity of that element in either of the sets involved in the intersection operation. Thus, for

$$A = \{a, b, c, c\} \text{ and } B = \{a, b, c, c, d, e\},$$

we have

$$A \cap B = \{a, b, c, c\}.$$

However, because our databases seek to preserve information regarding the origin of the hash values (i.e., which disk image they were imported from), our multiset intersection operation *adds* the multiplicities of the elements found in either set, essentially functioning as a multiset union operator for only elements that have some representation in both sets.

Therefore, in our multiset Jaccard similarity calculation, when we calculate the intersection of A and B we get 2 a's, 2 b's, and 4 c's.

$$A \cap B = \{a, a, b, b, c, c, c, c\}.$$

Because the choice of similarity metric is somewhat arbitrary, we do not believe this deviation has a significant impact on the outcome of the similarity calculations. Our multiset union operation follows the traditional definition; when we calculate the union of A and B we get 2 a's, 2 b's, 4 c's, 1 d, and 1 e.

$$A \cup B = \{a, a, b, b, c, c, c, c, d, e\}.$$

For our multiset Jaccard similarity calculation, the cardinality of the set intersection, numerator of our calculation, represents a matching hash each time it appears. In our example above, the cardinality of the intersection is 8 (2 a's + 2 b's + 4 c's). The denominator, the cardinality of the set union, will represent the total number of hashes in each set. For our example, the cardinality of the union is 10 (2 a's + 2 b's + 4 c's + 1 d + 1 e). Therefore, we calculate a multiset Jaccard similarity score of:

$$s(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{8}{10} = 0.8. \tag{2.5}$$

The set Jaccard similarity metric is a slight variation of the previously defined multiset Jaccard similarity. Both metrics apply a set intersection calculation; however, the set Jaccard similarity will only count a matching hash once—that is, each hash is treated as a type.

For example, we can see how the set similarity differs from the multiset using the same example of multisets as before:

$$A = \{a, b, c, c\} \text{ and } B = \{a, b, c, c, d, e\}.$$

However, representing A and B as sets, not multisets, we obtain:

$$A = \{a, b, c\} \text{ and } B = \{a, b, c, d, e\}.$$

When we calculate the intersection of A and B we get 1 a, 1 b, and 1 c.

$$A \cap B = \{a, b, c\}.$$

When we calculate the union of A and B we get 1 a, 1 b, 1 c, 1 d, and 1 e.

$$A \cup B = \{a, b, c, d, e\}.$$

For the set Jaccard similarity calculation, the cardinality of the set intersection, the numerator of our calculation, counts a matching hash one time no matter how many times it is represented in the database. In our example above, the total intersection is 3 (1 a + 1 b + 1 c). The denominator, the set union, will represent the total number of hashes (at most once) in each set. For our example, the total union is 5 (1 a + 1 b + 1 c + 1 d + 1 e). Therefore, we calculate a set Jaccard similarity score of:

$$s(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{3}{5} = 0.6. \tag{2.6}$$

To implement our multiset and set Jaccard similarity in software, we must manipulate the hash databases in such a way to emulate the behavior we expect. The hashdb tool (see Section 4.3) stores each hash along with a string representing the source of the hash (typically a file path and the offset into the source where the hash was acquired). We emulate the behavior of a multiset by making sure each hash has a unique source/offset associated with it, but then only look at the hash value part of the hash-origin pair when we are counting elements. We emulate the behavior of a set by doing the same thing but also making sure no hash value can occur more than once by creating our databases with the maximum number of hashes allowed equal to one.

It is important to note that the Jaccard similarity metrics are normalized based on the cardinality of the sets representing each image, which correlates to total disk image size. For two drives that are roughly the same size, this does not affect our results. However, if two drives are vastly different sizes, even a small similarity score may be meaningful. For example, if a thumb drive or mobile device contains files of interest matching a much larger hard drive, the overall size of the two drives will dwarf the importance of the relatively smaller matching sectors. This effect is mitigated somewhat by the set similarity (in contrast

to the multiset) since repeated sectors do not alter the score. However, focusing on types rather than tokens has a much greater impact on document similarity than sector similarity, since the diversity of sectors far exceeds the diversity of words.

2.2.4 Visualizing Similarity

To visualize our similarity calculations, we use an undirected graph to model our pair-wise similarity calculations. Treating each hard disk image as a node, or vertex, and similarity scores as edge weights we are able to visualize the connectedness of our images. Therefore, lines connecting any two images indicates a similarity score greater than zero where a thicker line represents a higher score. We use the Fruchterman Reingold [16] algorithm to optimize the distribution of nodes evenly and reflect symmetry. This allows us to easily identify clustering of similar images. Each graph is also color-coded based on the modularity class algorithm of Blondel *et al.* [17] which attempts to extract the community structure of large networks by grouping similar nodes. Figure 2.1 provides an example graph showing five nodes and edges as determined by similarity scores. A thicker line represents a higher similarity score.

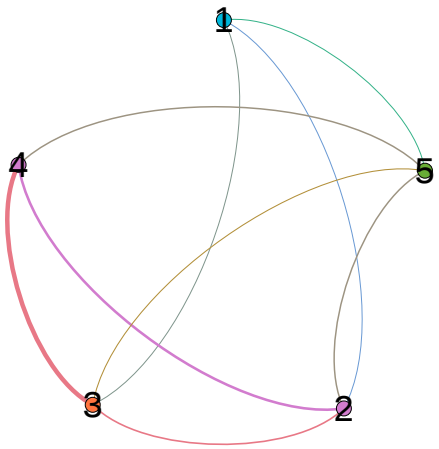


Figure 2.1: An example graph with five images represented as numbered nodes and connections indicating non-zero similarity score. A thicker lines represents a higher similarity score.

2.3 Bytewise Approximate Matching

Approximate matching is a general term for determining similarities between digital artifacts. In this section, we discuss fingerprints, cryptographic hashes, and bytewise approximate matching as a way to determine similarity between digital objects.

2.3.1 Fingerprints

A fingerprint, or signature, is an identifier for a larger object. It is a way of representing a larger digital object with a smaller identifier. A fingerprinting algorithm is a procedure that maps an arbitrarily large input to a smaller, fixed-length output. A fingerprint has the property that if two fingerprints are different, the corresponding objects are most likely different, and there is only a small probability that two different objects have the same fingerprint [18]. Creating fingerprints is a way to reduce the amount of storage necessary to identify a large amount of data. For example, storing the fingerprints of sectors, we can uniquely identify each block of a hard disk, and only have to work with the corresponding fingerprint, or hash database, as opposed to the larger original disk image.

2.3.2 Cryptographic Hashes

Cryptographic hash functions are an example of a fingerprint algorithm (see Section 2.3.1) and have been widely used to identify exact matches in digital forensic analysis. The Message Digest (MD5) algorithm, the hash function used in our analysis, takes as input a message of arbitrary length and produces as output a 128-bit "message digest" of the input. The MD5 message-digest algorithm is simple to implement, and provides a fixed-length output of an arbitrary-length object, which can then be compared with other fingerprints to identify exact matches quickly [19].

Cryptographic hash functions have three distinct properties applicable to digital forensics:

1. Pre-image resistance (one-way function): for any given message digest, it should be computationally infeasible to find a corresponding input which hashes to that digest.
2. Second pre-image resistance: it is computationally infeasible to find any second input which has the same digest as any specified input.
3. Collision resistance: it is computationally infeasible to find any two inputs which hash to the same output [20].

For digital forensics, pre-image resistance and collision resistance enable the ability to take an arbitrary length message as input and rely on its hash as a sufficiently unique representation of a digital artifact. Cryptographic hashes also exhibit what is called the "Avalanche Effect." A hash function displays the Avalanche Effect if a small change in the input to the hash function causes a large change to its message digest. This property is extremely beneficial when trying to obfuscate data (e.g., passwords in a shadow file), as two similar inputs have two very different hashes, but greatly reduces the ability to determine similarity between two digital objects solely by examining their hashes. When a file is changed only slightly, its digest changes drastically; this reduces our ability to find similar digital objects that vary even slightly. Consequently, an adversary can make minor changes to files and completely eliminate the possibility of detecting matches using techniques that rely on file-hashing. In addition, many file types contain header information that changes each time a user saves a file. This makes detecting similarity between files incredibly difficult as simply saving a file can result in a change to its hash (without changing its content). We attempt to overcome this limitation by hashing along sectors, which has the potential of detecting matching blocks of a file even if other blocks have changed (given the file's size is of sufficient length to stretch over multiple sectors).

Due to the unique properties of cryptographic hash functions, notably collision resistance, cryptographic hashes are commonly used to uniquely identify segments of digital media. Hashes can be used to identify entire hard disks, individual files, or equal-sized segments of a hard disk or file. While MD5 is known to have weaknesses as a cryptographic hash, its vulnerability stems from an attacker's ability to create collisions. Although collisions could potentially inflate similarity scores computed in our work, the motivation for this deception strategy is unclear, and we expect the effect to be minimal. We therefore use MD5 despite this issue because of its speed advantages [21].

2.3.3 Bytewise Approximate Matching

Determining if two drives are similar at a binary level falls into the approximate matching category called bytewise matching. Bytewise matching is in reference to the fact that the similarity decision relies solely on the sequence of bits that make up the digital object.

Bytewise matching works in two parts:

1. A compressed representation of the digital object is created (in this case a collection of MD5 hashes of equal-sized sectors), and
2. A similarity score is calculated from these representations.

Using equal-sized sectors enables digital forensic practitioners to segment a dataset and uniquely identify each block utilizing a cryptographic hash. This allows for comparisons to be made across large diverse sets of data by comparing hashed sectors [22].

2.3.4 Block Types

There are three ways to describe types of blocks: unique, distinct, and probative. The definitions of their use follow.

A unique block is a data block that appears only once. A truly unique block provides little benefit to a forensic analyst looking for matches—as there are none.

A distinct block is defined as "a block of data that will not arise by chance more than once" [23]. This implies that a distinct block is created by a random process (e.g., JPEG format pictures taken of nature). It is important to note while it is possible to determine if a block is not distinct, it is impossible to determine the contrary, that a block is distinct, with certainty. However, Garfinkel posits two hypotheses of distinct blocks important for forensic analysis:

1. Distinct Block Hypothesis #1: If a block of data from a file is distinct, then a copy of that block found on a data storage device is evidence that the file was once present.
2. Distinct Block Hypothesis #2: If a file is known to have been manufactured using some high-entropy process, and if the blocks of that file are shown to be distinct throughout a large and representative corpus, then those blocks can be treated as if they are distinct [23].

Finally, a probative block is an extension of a distinct block used to "describe data that conveys a high probability that an entire file (or a file based on the entire target file) was once present" [24]. Garfinkel *et al.* changed terminology from distinct to probative after determining that seemingly distinct blocks are often blocks that repeat infrequently without necessarily providing evidence of the presence of the rest of the file; a fact discovered when blocks were compared to a larger corpus. We observe this phenomenon in our analysis as

we look at very large datasets containing many images.

CHAPTER 3:
Previous Work

Our research begins with the culmination of work conducted in very different areas. To cover each topic, four broad themes will be used to illustrate the previous work preceding our ability to conduct large-scale similarity detection:

1. Foundations of digital forensic matching techniques
2. Forensic applications of cryptographic hashing
3. Context-sensitive matching, and
4. Block hash matching

This chapter begins with the basic concepts of similarity as a mathematical set and how we can apply efficient string fingerprinting to digital objects. From there, we examine the evolution of cryptographic hashing techniques to identify various segments of digital data. Finally, we conclude with specific forensic techniques for matching digital objects.

3.1 Foundations of Forensic Matching Techniques

Determining the similarity between two digital objects requires a repeatable method for determining and quantifying similarity. In this section, we begin with the foundations of mathematical sets, uniquely identifying digital data, efficient string fingerprinting, and the application of set theory to digital documents.

3.1.1 Jaccard Similarity

Paul Jaccard [25] first considered similarity, and dissimilarity, during his research of the geographic locations of common and rare Alpine flora. In an effort to understand the historical causes of the location of such flora, Jaccard divided the landscape into districts based on similar and dissimilar characteristics of the topographical and petrological landscape. He then began examining the ratio of species common to two districts to the total number of species collected. This ratio, referred to at the time as the coefficient of community, is the basis for what is now called the Jaccard similarity. Jaccard's primary contribution was to

express similarity as a relationship of two (unordered) sets. Specifically, the Jaccard coefficient measures similarity between finite sample sets, and is defined as the cardinality of the intersection divided by the cardinality of the union of the sample sets (see Section 2.2.3).

3.1.2 One-Way Hash Functions

Merkle [26] developed the idea of a one-way hash function as a means to authenticate a large data field with a smaller data field. In his research to create a public key cryptography system, he devised a one-way hash function that could take any size input, produce a fixed-size output, and be computationally infeasible to recreate the input given the output. Hashes provide an efficient way to take an arbitrary amount of input and provide a fixed output reducing the space necessary to uniquely identify and store digital media. This construction was used in the design of many popular hash algorithms including MD5, SHA1, and SHA2.

3.1.3 Rabin Fingerprinting

In 1981, Rabin [27] developed a simple real-time string matching algorithm and a procedure for ensuring the integrity of files. The method uses irreducible polynomials to fingerprint bit streams and is highly efficient and reliable for every input. The Rabin fingerprint has been the foundation for new techniques since its development, and can be viewed as a way to efficiently detect similar, but not identical objects. Using a sliding window, the algorithm decides where to begin and end based on the data in the current window (forming what is called the Rabin fingerprint). Using a sliding window and removing the rigidity of predefining equal-sized sectors and offsets eliminates the limitation of such similarity algorithm's ability to identify similarity when data is added to the beginning of a file thus shifting sector alignment. However, this method has been shown to produce high numbers of false-positives [28].

3.1.4 Document Similarity

Broder's [13] application of set-based similarity to documents revolutionized text document search. The concept of resemblance (see Section 2.2.2) in similarity detection applies very well to finding textually similar documents. Broder *et al.* [29] applied this concept to the World Wide Web (pulled from an AltaVista spider run at the time), building a clustering of all the documents that were syntactically similar. The general technique depends on the

ability to extract a set of features from objects. Viewing each document as a sequence of words, Broder associated with every document a subsequence of words, called shingles, of size 10. He then used a variation of Rabin's technique to fingerprint each shingle. He then compared the fingerprints for each pair of documents to see if they exceed a threshold of resemblance, and combined the pairs of similar documents to make clusters of similar documents. This technique is not limited to text documents. Given any technique that extracts a set of features from an object, we can measure the similarity of any two objects or cluster the sets of similar objects from a large number of objects.

3.2 Forensic Applications of Cryptographic Hashing

Cryptographic hashes (see Section 2.3.2) are a common way digital forensic practitioners uniquely identify digital objects. Due to the nature of hashing algorithms, if two digital objects are hashed and compute to the same digest, the probability that the original two objects are the same is extremely high. This section examines the evolution of the use of hash functions for forensic analysis.

3.2.1 File Hashing

Merkle's development of a one-way hash function (see Section 3.1.2) devised a means to fingerprint large amounts of digital data for integrity purposes. The application of a one-way hash as a means to authenticate large amounts of data with a smaller identifier created the possibility for forensic analysts to begin to fingerprint data and develop matching techniques. The use of hashes are relied upon heavily in legal cases. In December 2006, the Federal Rules of Civil Procedure were amended to establish new standards concerning the preservation and discovery of electronically stored information (ESI)—particularly from the use of hash functions [30]. Forensic examiners have used hash values throughout the forensics process. First, hashes were used to fingerprint original data. Before, during, and after an examination, the data would be hashed and each hash would have to match to confirm the integrity of the original data. Second, hashes were used to filter out files that were of no interest, such as operating system files, and to identify files of particular interest. For this purpose, NIST maintains a database of hashes for common operating systems and applications [31]. However, file hashing is highly vulnerable to the Avalanche Effect. A slight modification to a file, intentional or not, will result in a completely different hash.

File hashing is also dependent on the filesystem to locate the beginning and end of files. If digital media is corrupted or altered to hide user data from the operating system, file hashing may not detect these files.

3.2.2 Piecewise Hashing

Piecewise hashing, originally developed by Nick Harbour while at the Department of Defense (DOD) Cyber Crime Center for dcfldd [32], segments an entire file into discrete-sized pieces and computes a hash for each piece rather than a single hash for the entire file. The technique was originally developed to mitigate errors during forensic imaging. If an error occurred, only one of the piecewise hashes would be invalidated, and the integrity of the remainder of the hashes would be assured [33]. However, similar to file hashing, an adequate representation of the file system is necessary to break each file into blocks and may not be possible given the state of the imaged digital media. Piecewise hashing is the basis for many of the tools and techniques used by digital practitioners.

3.3 Context-Sensitive Matching

Kornblum developed [34] Context Triggered Piecewise Hashing (CTPH) as a combination of Harbour's piecewise hashing technique and the rolling hash of a Rabin fingerprint. CTPH can match inputs that have homologies—inputs containing sequences of identical bytes in the same order, but with bytes in between these sequences that may be different in both content and length. The proof of concept for CTPH was implemented in ssdeep [35], which was able to detect similarities in a document given insertions, deletions, and modifications to the aesthetics of the file (e.g., font size, font color, spacing). However, as each input may need to be processed multiple times to determine the similarity, the performance of ssdeep suffers compared to traditional hashing techniques.

Roussev [28] also developed a technique for similarity based off a statistical approach for selecting fingerprinting features as opposed to Rabin fingerprinting. The technique uses statistically-improbable features to generate similarity digests by selecting features that are least likely to occur in other data objects by chance. The proof of concept was implemented in sdhash [36]. By focusing on statistically-improbable features, as opposed to randomized feature selection, sdhash reduces the false-positives rate of Rabin fingerprinting. However,

without a way to sort the bloom filters of which the signature is comprised, the performance suffers and is not scalable.

3.4 Block Hash Matching

Block hashing is the application of cryptographic hashes to smaller-than-file size segments of data. Its application has evolved from blocks of files to blocks as sectors. A description of these techniques follows.

3.4.1 Bulk Analysis

Typical digital forensic analysis relies on how a human interacts with a computer (e.g., creation of images, videos, documents) due to its ease of understanding and transition to the courtroom as evidence. A complementary approach called bulk data analysis was examined by Garfinkel [37] and disregards the filesystem and focuses on content. This allows for analysis to be agnostic to specific computer systems, file systems, and file types. Garfinkel used the open source forensic tool bulk_extractor (see Section 4.3) to show its particular effectiveness in triage, especially in cases that involve multiple pieces of digital evidence. The tool bulk_extractor processes digital data from beginning to end and is not concerned with the structure of the filesystem.

3.4.2 Hash-Based Carving

Hash-based carving is a technique based on matching hashes of physical sectors of known hashes to target files (e.g., known illicit files) [38]. As a proof of concept, Garfinkel [39] used fragments of text from the DFRWS 2006 Carving Challenge to create Google query terms to find the actual source documents used on the Internet. He then broke these documents into 512-byte fragments, hashed each block, and searched the Challenge for 512-byte sectors with matching hash codes. Using this technique, Garfinkel identified three of the documents in the challenge image, including a Microsoft Word file that was divided into three fragments.

Garfinkel *et al.* [23] conducted additional research segmenting digital objects into blocks instead of looking at the whole object. Breaking target files into 4096-byte blocks in order to match the sector size used on most current hard drives. Each fragment is called a block

with its corresponding hash known as a block hash. In the same process as searching for complete target files, 4096-byte sectors are read, hashed, and compared to block hashes of target files for matches. However, the implementation did not scale to large systems and underestimated the difficulty of determining the importance of a block hash match. To overcome the efficiency issues, hashdb (see Section 4.3) was created as a custom database to handle hash-based carving.

Young *et al.* [40] further examined the block hash matching approach by searching disk sectors for distinct file blocks, rather than searching the file system for distinct files. This method is no longer concerned with the file system or file type and includes all portions of the digital media. Breaking the sectors into equal-sized partitions, often 512-byte or 4096-byte sizes, allows detection of target files even if other sections have been changed. Choosing a suitable block size is a balance between a small enough granularity to correctly detect matches and the time and space necessary to maintain and process each hashed sector. However, alignment issues arise when data is inserted at the beginning of a file resulting in different sector hashes as the data shifts across sectors. As file header data grows or changes, resulting sector offsets change and so do their corresponding hashes.

Garfinkel and McCarrin [24] furthered the technique called hash-based carving which compares the hashes of specific data blocks instead of entire files. Evaluating data blocks allows for more flexibility in detection of incomplete, fragmented, or modified files depending on the size of the block chosen. Hash-based carving identifies hash matches as a result of a variety of scenarios, including:

1. A copy of an intact target file is present on the searched media. Because hash-based carving is file system agnostic, it makes no difference if the file is allocated, deleted, or in free space.
2. A copy of the target file might have been placed on the searched media at some time in the past and later deleted and partially overwritten. In this case, there may be small fragments of a target file on searched media that are probative.
3. A file that has many sectors in common with the target file may be on the searched media. In this case, hash-based carving will identify the blocks shared between the two similar files.
4. A target file may be embedded in a larger carrying file, provided that the file is

embedded on an even sector boundary. For example, Microsoft's ".doc" format embeds objects such as JPEG files on 512-byte boundaries, but the ".docx" format does not. [24]

Combining hash-based carving and the hash database storage tool hashdb greatly improves performance by sorting the hashes for faster lookup times. However, segmenting files into data blocks also increases the number of hashes to be computed and stored potentially limiting its effectiveness with very large amounts of data.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Methodology

To examine similarity amongst a large set of disparate data sources, we selected a test set of 100 drive images, hashed each 512-byte sector of each image, and stored the resulting hashes in 100 databases. We then performed five experiments designed to collect information regarding the sector-level similarities between the drives in this set, with the goal of understanding which high-level characteristics of the drives corresponded most closely to our similarity measures.

4.1 Dataset

Our experiment worked with a randomly chosen sample of 100 drives taken from the Real Data Corpus (RDC) [41]. The RDC is a collection of disk images taken from various storage devices purchased from the secondary market. At the time of this writing, the full RDC consists of over 60 terabytes of images of extracted from hard drives, cell phones, and flash drives extracted from all countries except the United States.

To create our sample set, we randomly chose 100 disk images from the RDC (see Appendix A.1 for a list of disk image filenames). The purpose of this reduced dataset is to make our analysis computationally feasible while providing a large-enough sample to be representative of the corpus. The primary difficulty for our approach is the requirement for a pair-wise similarity calculation. Following the classic solution to the handshake problem, the numbers of calculations necessary for n images is given by:

$$\frac{n(n-1)}{2}. \tag{4.1}$$

To compute similarities for all 3,000 images in the full RDC would require 4,498,500 comparisons. Given time limitations to processing time on Hamming (see Section 4.2) and deadline constraints on the current work, we chose a subset of 100 images. This reduced the pair-wise similarity requirement to only 4,950 comparisons.

4.2 Hardware

We ran the majority of our analysis using the Hamming compute cluster in the High Performance Computing Center (HPC) at Naval Postgraduate School (NPS). At the time of processing, Hamming contained 3,858 computing cores, 17.75 terabytes of memory, and 22.7 terabytes of hard disk space. HPC on Hamming allowed the parallel execution of many analysis tasks. Much of our methodology is embarrassingly, or pleasingly, parallel and requires very little effort to separate into parallel tasks. As a result, total wall-clock time was reduced by performing the bulk of our computations in parallel on Hamming.

4.3 Software

Our methodology depended on four open source tools: we relied on bulk_extractor and hashdb for image processing and hash database manipulation, Gephi for graph visualization, and SectorScope for analysis of files and sector content represented by matching hashes.

4.3.1 Bulk Extractor

Before selecting our sample set, we used the open source forensic tool bulk_extractor to create hash databases of 512-byte blocks for each image in the RDC. Bulk_extractor's general strategy is to scan a disk image and extract specific features without parsing the file system or file system structures. Ignoring file systems allows bulk_extractor to be used to process any source of digital media and also improves processing speed by proceeding linearly through the scanned media, thus avoiding disk seeks. The tool scans each 512-byte block, hashes each block, and uses the hashdb scanner to import the hash into a hash database [42]. Each image's hash database could then be scanned and queried using the tool hashdb.

4.3.2 Hashdb

We used hashdb for storage of hashes created from our images. The tool is designed to store upward of one billion MD5 hashes and can be populated and queried with the corresponding bulk_extractor scanner, which can be run against both files or whole disk images. Built-in options enable the scanner to import the hash of every (non-overlapping) 512-byte sector

on a disk image. It also allows databases to be queried and merged, and supports operations for set intersection and union, which were useful for calculating our similarity metrics [43].

4.3.3 Gephi

Gephi is an open-source tool used to explore and visualize graphs. Users can input graph data in various graph formats or as a comma-separated values (CSV) file. Layout algorithms can then be applied to give shape to the graph enabling full control of the layout. Applying filters to the graph (e.g., edge weight thresholds, minimum connection requirements) enables exploration of the graph's properties. Gephi also provides a statistics and metrics framework for social network analysis [44]. In our visualizations, nodes represent disk images and edge weights correspond to similarity scores where a link between nodes indicates a similarity score greater than zero.

4.3.4 SectorScope

SectorScope is a visualization tool used for showing how matched blocks are distributed across a media image. Matches are obtained by scanning the image for previously identified blocks against a pre-populated database containing the hashes of these blocks using the bulk_extractor hashdb scanner [45]. SectorScope displays the frequency of matches as a histogram at given offsets of the disk. Users can highlight portions of the media to view sectors of interest on the chart. A source table provides the source ID, the percent of this source that was matched by the scan, the number of matches matched by the scan, the size the source file matched, and its repository (see Figure 4.1). SectorScope allowed us to import a hash database of hashes that we know to appear on multiple disk images in our collection, locate these sectors on a single disk image, and inspect the content of these sectors and their neighbors (as a hexdump).

4.4 Overview of Methodology

We designed five experiments to identify sector-level similarities between our drives and understand the relationship between these similarities and high-level features of our dataset, such as common files, operating system version, file system or region of acquisition. A high-level description of these experiments is as follows:

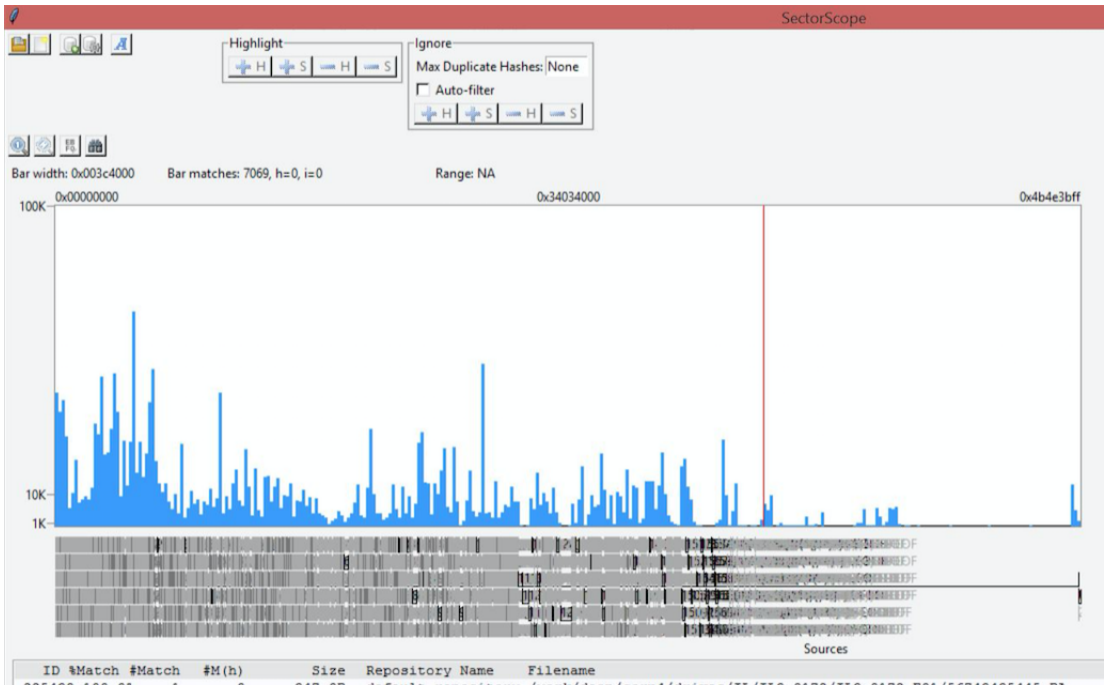


Figure 4.1: SectorScope shows the number of sectors matching as a histogram as you move through the sectors of the hard disk image along the x-axis. This sample output was taken after scanning image 7 in our dataset against our hash database of known duplicate hashes. The higher frequency sectors represent hashes that duplicate the most. Highlighting an area along the graph allows the user to zoom in and view the sectors with more detail as well as view the hex/ascii representation of the highlighted section.

1. Compute pair-wise similarity scores and visualize the results
2. Test for a common set of hashes across the dataset
3. Find frequency and distribution of common hashes
4. Compare individual drive images to common hash set
5. Test for common set of hashes by communities

We describe each experiment in detail in the remainder of this chapter.

4.4.1 Experiment 1: Similarity Scores and Visualization

Our first experiment consisted of calculating similarity scores between each image in our dataset. This process required creating hash databases of each image, performing set in-

tersection and union operations between hash databases, and calculating each pair-wise similarity score using two metrics (see Section 5.1 for a description of similarity metrics). Once all comparisons were completed, we constructed a graph to visualize their interrelationships by representing disk images as nodes and similarity scores as weighted edges between them.

Hashing Individual Disk Images

We scanned each image in the full RDC with bulk_extractor using the hashdb scanner to hash each 512-byte block, and import the results into a hashdb hash database. Creating a separate hash database for each disk image in the corpus allowed us to treat each image as either a set or multiset of hashes (depending on the requirements of the task) and to perform the intersection and union operations necessary for each similarity score. We chose sectors of 512 bytes because this is the smallest addressable unit of most block devices, and file systems tend to align files on sector boundaries for performance reasons [24]. Consequently, we expect that our sector hashes will match in cases where identical or similar files are stored on different disk images at different offsets, regardless of the partition alignment or fragmentation. The choice of 512-byte sectors also maintains backward compatibility with newer, advanced format drives using 4096-byte sector sizes, which will also be aligned to 512-byte boundaries (a 4096-byte sector is eight 512-byte sectors).

We processed each image in parallel on the Hamming HPC cluster (see Section 4.2). Due to limitations on the number of active running jobs at a single time, we ran between 50 and 60 jobs simultaneously. Because hashdb maps memory during operation, a strategy that can be problematic on cluster file systems such as lustre, each raw image had to be transferred to a single compute node for each array job to be processed correctly. Transferring each image to the compute node increased the communication cost of the jobs. However, we incurred this penalty only once for each disk image, after which we discarded the local copy and retained only the resulting hash database and logs necessary for analysis. These results require approximately $1/7$ of the storage space of the original data (to include log and history files), and can be compressed to further reduce their footprint (requiring roughly $1/16$ their original size).

Similarity Metrics

Two metrics were chosen to measure the similarity between each image: a multiset Jaccard similarity and a set Jaccard similarity ranging from zero to one where a score near 1 indicates high similarity and a score of 0 indicates no matching sectors (precise formulas for each score are provided in Section 2.2.3).

Multiset Jaccard Similarity

To calculate the multiset Jaccard similarity metric, we create two new hash databases for each pair: one of these databases represents the intersection of the two multisets of hashes, and therefore contains only hashes that appear in both; the other represents multisets and contains all hashes that appear in either multiset. We use the hashdb `intersect_hash` command to create the former. We can then use the hashdb `size` command to find the cardinality of the intersection of the two images. This becomes our numerator in the similarity calculation. We create a new hash database containing the union of the two databases using the hashdb `add_multiple` command and again find its cardinality with the size command. Once the cardinality of the two multisets is calculated, simply dividing the intersection by the union results in our multiset Jaccard similarity score. The sequence and explanation of commands follows.

1. To create a new database <multiset_intersection> containing the hashes of the intersection, we run:
   `hashdb intersect_hash <hashdb1> <hashdb2> <multiset_intersection>`
2. To create a new database <multiset_union> containing the hashes of the union, we run:
   `hashdb add_multiple <hashdb1> <hashdb2> <multiset_union>`
3. To find the cardinality of the intersection, we run:
   `hashdb size <multiset_intersection>`
4. To find the cardinality of the union, we run:
   `hashdb size <multiset_union>`
5. Lastly, we calculate the multiset Jaccard similarity metric by:
   `Multiset Jaccard similarity = (result of step 3)/(result of step 4)`

Set Jaccard Similarity

To calculate the set Jaccard similarity utilizing hashdb we follow a very similar process as the multiset Jaccard similarity and create two new hash databases. However, instead of allowing hashdb to create our new databases as in the previous commands, we first create two databases with the `-m 1` option to set the maximum distinct hash count of each database to one. This limits the number of hashes inserted into the database to only one and will not permit the insertion of additional hashes with the same value. Once these databases are created, we follow the same steps as before. The sequence and explanation of commands follows.

1. To create the hash database <set_intersection> with a maximum hash count set to one, we run:
   `hashdb create -m 1 -p 512 <set_intersection>`
2. To create the hash database <set_union> with a maximum hash count set to one, we run:
   `hashdb create -m 1 -p 512 <set_union>`
3. Calculate the set intersection of the two hash databases inserting the results into our <set_intersection> database
   `hashdb intersect_hash <hashdb1> <hashdb2> <set_intersection>`
4. Calculate the set union of the two hash databases inserting the results into our <set_union> database, we run:
   `hashdb add_multiple <hashdb1> <hashdb2> <set_union>`
5. To find the cardinality of the intersection, we run:
   `hashdb size <set_intersection>`
6. To find the cardinality of the union, we run:
   `hashdb size <set_union>`
7. Lastly, we calculate the set Jaccard similarity metric by:
   `Jaccard similarity = (result of step 5) / (result of step 6)`

Visualization

We graphed the results of both the multiset and set Jaccard similarity calculations to better understand the relationships between disk images in our set. Treating our disk images as nodes connected by edges with edge weights equal to the similarity score calculated

between each pair of images, we were able to visualize our dataset as an undirected graph (see Section 2.2.4) In addition, we created graphs with metadata overlays for both the country of origin and partition type of each image (see Figure 2.1).

4.4.2 Experiment 2: Duplicating Sector Hashes

Our second experiment tested whether we could isolate a set of sector hashes common to all images in our dataset with non-zero similarity scores. This effort proceeded in two phases. First, since any set of hashes common to all connected images in our graph must be found on a fully-connected core of this graph, we used the k-core filter in Gephi to determine the largest fully-connected component of the graph. Second, we used hashdb to find the intersection of all sets of hashes corresponding to the disk images in the fully-connected component. However, because hashdb does not allow the intersection of more than two hash databases at a time, we recursively intersected pairs of hash databases, then pairs of the resulting intersections, and so on such that each successive iteration halved the number of databases remaining to be intersected to arrive at a single database containing the intersection of all images represented in the fully-connected component (see Figure 4.2). In addition, we also examined the filesystem of the images with sector hash matches for exact matches (similarity scores equal to one) and images which have no connections with any other images in our dataset looking for interesting cases.

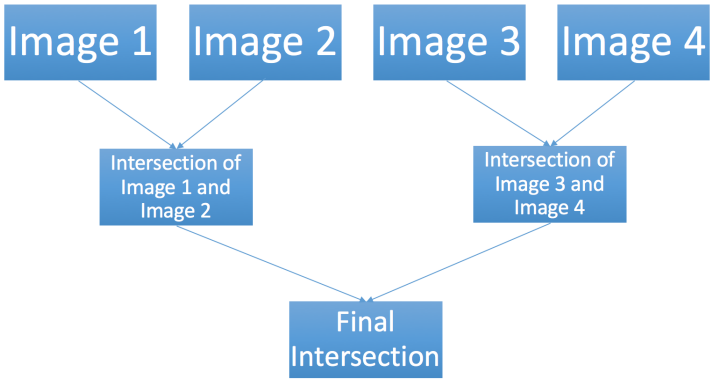


Figure 4.2: Example workflow for recursive intersection of four hash databases created from disk images.

4.4.3 Experiment 3: Frequency of Common Hashes

Our third experiment consisted of determining the frequency and distribution of repeating hashes in our dataset. Examining the most frequently occurring sector-level matches will provide context as to what high-level similarities are being represented by the matches. It is possible that a small number of sectors are responsible for the majority of matches across disk images as a result of a few commonly-occurring sectors. The file or type of file represented by these sectors may then occur frequently across our disk images. Alternatively, the matches may be produced across many sectors that all appear with about the same frequency and thus contribute equally to the matching. In this case, the sources may result from common file headers or some low-entropy data structure that do not correspond to meaningful content.

Most Frequently Occurring Hashes

To study the frequency with which hashes occur on multiple drives (and thus contribute to edge weights in our graph) we constructed an aggregate database containing all such hashes. Due to memory limitations, we cannot operate on the entire 100 images at the same time as the size of hash databases becomes too large to manipulate efficiently. Empirically, we found that our hardware, when operating on datasets larger than 1 billion hashes, began to produce frequent page faults, greatly reducing the speed of database operations such as merges and intersections, which require iterating through the entire set. Because of this constraint, we could not follow the straightforward approach of merging all hashes into a single database, then discarding hashes that appear on only one drive. Fortunately, because most hashes occur only once, the set of hashes that appear on more than one disk image is significantly smaller than the total set of hashes. For this reason, we developed an alternative strategy for building our database in which we iteratively identified and discarded hashes that appear on only one disk image.

Our method proceeded in three phases:

In phase 1, we began by creating, for each disk image in our test set, a database containing exactly one of each distinct sector hash value found on the image (i.e., hashes of duplicate sectors appearing on the same disk image are discarded). Next, we recursively merged databases until the initial set was reduced to six databases of 2-4 hundred million hashes each.

For ease of reference, we labeled the set of these six databases D, where $D = \{d_1, d_2, d_3, d_4, d_5, d_6\}$.

Because the initial databases contained no more than one of each type of hash, the number of times a hash appears in a given database, d_i, corresponds exactly to the number of databases merged to create d_i that contained that hash. Consequently, it also represents the count of the number of disk images on which that hash appeared in the subset of disk images from which d_i was created.

Hashes that appear more than once in our databases are certain to affect our similarity graph. Hashes that appear exactly once are either irrelevant because they are globally unique within the dataset, or locally unique hashes that have matches we have not yet discovered.

We separate these two groups of hashes by splitting each database into two parts, labeled d_{ia} and d_{ib}, for each $i \in \{1, 2, 3, 4, 5, 6\}$ where d_{ia} contains all hashes that appeared more than once in d_i and d_{ib} contains all hashes that appeared exactly once in d_i.

Let $A = \{d_{1a}, d_{2a}, d_{3a}, d_{4a}, d_{5a}, d_{6a}\}$ and $B = \{d_{1b}, d_{2b}, d_{3b}, d_{4b}, d_{5b}, d_{6b}\}$. All hashes in any database in A are contributing to the edge weight of our similarity graph. We merge all these databases to create the hash database matches_1.hdb.

In phase 2, we focus on finding hashes in the databases in B that were locally unique in the corresponding database in D, but match hashes in other databases in B to which they have not yet been compared. Since the databases in B are too large to merge into a single database, we proceed instead by performing pairwise intersections of all six databases in B to yield 15 databases containing any common hashes between the databases being compared. We then merge these to create matches_2.hdb.

Finally, in phase 3, we focus on finding hashes in the databases in B that were locally unique in the corresponding database in D, but match hashes in other databases in A to which they have not yet been compared. Since all databases in A were merged into the matches_1.hdb we intersect all databases in B with matches_1.hdb and merge the resulting six databases to produce matches_3.hdb.

We then merge matches_1.hdb, matches_2.hdb and matches_3.hdb to produce a single database, which we call all_matches.hdb. This database contains every type of hash that

appeared on more than one disk image in our original set, and the number of times it appears corresponds with the number of images on which it was found.

4.4.4 Experiment 4: Repeating Sector Hashes

In our fourth experiment, we scanned individual images against the all_matches.hdb hash database created in Experiment 3 (see Section 4.4.3) to find and inspect the content of matching sector hashes across disk images. Graphing disk images based on similarity score visualizes connectedness based on similarity scores (a score greater than zero will be indicated by a edge drawn between the two images—see Figure 2.1). However, the property of being fully-connected does not imply a non-empty intersection. For example, the three sets:

$$A = \{a, b\}, B = \{b, c\}, C = \{c, a\},$$

are fully-connected (have similarity scores greater than zero): A and B share element b, B and C share element c, and A and C share element a. However, there is no single element which is represented in each set (i.e., $|A \cap B \cap C|$ is zero).

To find matching sectors across our dataset, we began by examining the SectorScope interactive histogram which shows the count of sector-matches scanned from the individual image against the database of known hashes. The graphical display represents sectors of the disk image on the x-axis and the count of matching sectors on the y-axis (see Figure 4.1). We can identify areas of interest based on the count of matching sectors. We zoom into areas showing high concentrations of sector matches by highlighting the area on the histogram. The tool SectorScope calculates the histogram bin size based on the pixel size of a given zoom level and can be difficult to identify a specific number of exact matches across images given an image with hundreds of millions of sectors due to the horizontal space necessary to represent such a large image at such a fine granularity. Therefore, we choose where to zoom in on the histograms based on the highest number of sectors at the furthest zoom level. Once we have zoomed into the lowest level possible and have identified a contiguous sequence of matching sectors, we then examine the hex dump of each sector match and the sectors preceding it to attempt to correlate individual sector hashes with their corresponding individual files. However, this process is dependent on the image containing a valid file

system to produce an accurate representation of files from sector hashes. The identified matching sectors from SectorScope were then cross-referenced with the files occupying those sectors in Autopsy when possible.

4.4.5 Experiment 5: Community Identification and Detection

In our fifth experiment, we grouped the images based on communities (i.e., operating systems) and tested whether matching sector hashes were able to identify communities of interest within our dataset. We chose images based off of matching sector hashes from SectorScope scans against individual drives from Experiment 4 (see Section 4.4.4). We determined the operating system of each image by examining its filesystem, if available, and its various system files. If its filesystem was not intact, we attempted to identify the operating system by searching its hex dump for various strings related to the different operating system versions.

We first computed the multiset and set intersection of each image within its respective community to determine the duplicate sector hashes shared by operating system. We then performed pair-wise intersections of each community to examine the resulting multiset and set of matching hashes between each community. If the result of the pair-wise intersection between multisets decreased, we could determine there were differences between communities. However, if the multiset intersections increased, we could determine the communities share sector-level hashes. Because some of the operating systems were unable to be determined, an increasing multiset may help identify previously unknown operating systems by testing their multiset intersections against each community.

CHAPTER 5:
Results

The results of each of our five experiments contributed to our understanding of the sector-level similarities in our dataset, and how these relate to higher-level relationships that may be of interest to analysts. The overarching trajectory of our analysis is as follows: once we succeeded in converting each of the drives in our dataset into a set of hashes stored in a hashdb instance and performing our all-to-all pair-wise comparison of these sets (Experiment 1), we proceeded to look for a common set of hashes present throughout the similar disk images (Experiment 2) that might explain the very highly connected graphs that resulted from the pair-wise comparison. Upon discovering that no such set existed, we examined the frequency and distribution of the most common hashes in the whole set (Experiment 3) with the aim of determining whether we could find a relatively small set of hashes that—if not common to all the similar drives—at least played a dominant role in producing most of the matches. Again, we hoped that this analysis would lead us to a small core set of sectors that we could examine to explain the similarities between the majority of the disk images. However, the matches were not highly concentrated in most cases, but were distributed across many hashes that matched just a few times. Instead, we moved from matches based on similarity scores to matches at the disk-level view (Experiment 4) and came up with our theory that the commonality was based on operating system version, which we determined was sufficient in explaining higher-level similarities in our dataset (Experiment 5). This chapter presents a detailed description of the analysis of each of our experiments. Accordingly, it is divided into sections corresponding to each, as follows:

1. Experiment 1: Summary statistics of similarity scores and visualization
2. Experiment 2: Common set of hashes across dataset
3. Experiment 3: Frequency and distribution of common hashes
4. Experiment 4: Matching sector-level hashes across dataset
5. Experiment 5: Common set of hashes by communities

5.1 Experiment 1: Summary Statistics of Similarity Scores and Visualization

We begin our discussion of the results of experiment 1 by examining summary statistics for our multiset and set Jaccard similarity metrics. We then visualize the results as graphs to better understand the connectedness of our dataset.

5.1.1 Summary Statistics

Examining the histograms for multiset (Figure 5.1) and set (Figure 5.2) Jaccard similarity scores, we can see a majority of the scores are less than 0.01 for both the multiset (2,189) and set (2,572) Jaccard similarity scores. We can also see that there are only 176 multiset and 44 set similarity scores greater than 0.10. Given such small scores, we can begin to understand how small the number of intersecting sector hashes are between images as compared to the total number of hashes of each image combined. Also, a high number of similarity scores are zero (1,832) showing not all images in our dataset contain shared sectors with other images.

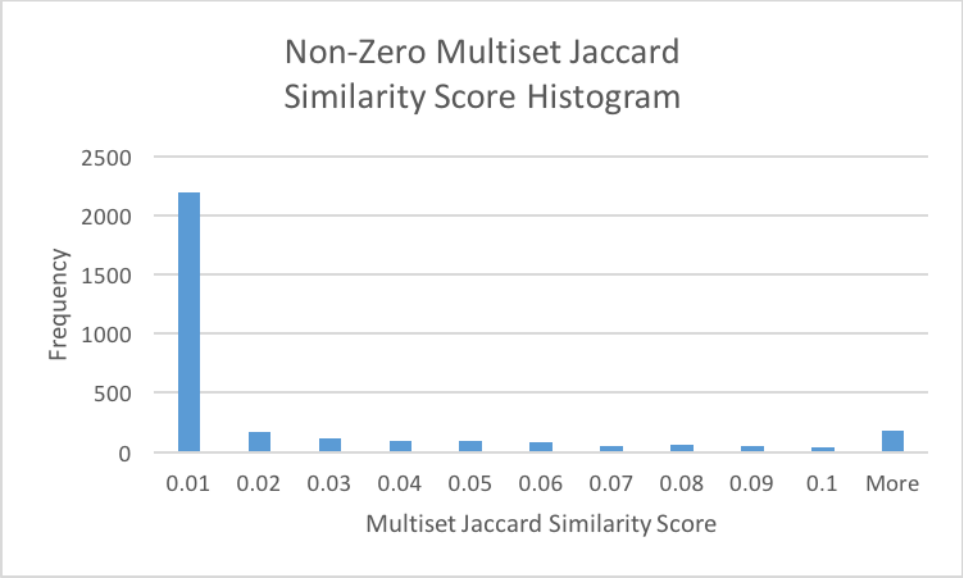


Figure 5.1: This histogram shows the frequency of non-zero multiset Jaccard similarity scores calculated for our 100 pair-wise calculations. Note the high number of scores less than 0.01 and fairly even distribution between bins greater than 0.01.

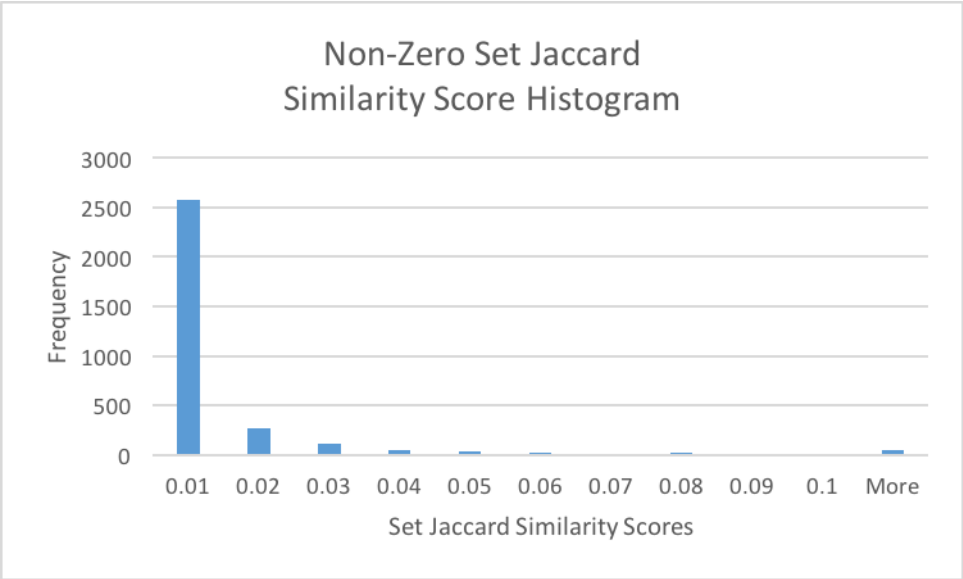


Figure 5.2: This histogram shows the frequency of non-zero set Jaccard similarity scores calculated for our 100 pair-wise calculations. Although the scores are lower than the multiset similarity scores overall, the distribution is similar.

As shown by the summary statistics in Table 5.1, the multiset and set Jaccard similarity scores span the full range of possible values between zero and one, but are strongly concentrated toward the lower end of the spectrum. The multiset mean is 0.0137022 compared to 0.0053746 for the set mean, and the multiset median is 0.0000145 compared to 0.0000005 for the set median. Treating each hash as a type in our set calculation resulted in overall lower values when compared to the multiset calculation, where each hash is treated as a token, due to the counting of each hash only once. We can also see that our dataset contains at least one exact match (similarity score equal to one) and the most frequently appearing similarity score was zero for both the multiset and set Jaccard similarity scores (1,832).

Differentiating the two similarity scores based on our summary statistics does not provide conclusive evidence that either metric is capturing the similarity between disk images better than the other, but the multiset similarity does seem to provide more information for distinguishing between the strength of the matches (not as many scores are clumped together in the 0.01 bin) and shows a more even distribution among bins.

	Multiset Jaccard Similarity	Set Jaccard Similarity
Minimum	0.0000000	0.0000000
Quartile 1	0.0000000	0.0000000
Quartile 2	0.0000145	0.0000005
Quartile 3	0.0064589	0.0015949
Quartile 4	1.0000000	1.0000000
Maximum	1.0000000	1.0000000
Mean	0.0137022	0.0053746
Median	0.0000145	0.0000005
Mode	0.0000000	0.0000000

Table 5.1: Summary statistics calculated for our multiset and set Jaccard similarity metrics.

5.1.2 Graphs

We visualize the similarities by constructing graphs representing the images as nodes (with corresponding country of origin) with the edge weights equal to the multiset and set Jaccard similarity metrics (shown in Figure 5.3a and Figure 5.3b). After applying the Fruchterman Reingold algorithm [16] and color-coding based on the modularity class algorithm of Blondel *et al.* [17], we made three immediate observations:

1. A large majority of the images were highly connected among each other
2. A small number of the images had no connections
3. A small number of the images were tightly connected with only each other

Israel (IL)	30
China (CN)	22
Thailand (TH)	13
Bangladesh (BD)	8
Pakistan (PK)	7
Hungary (HU)	7
United Arab Emirates (AE)	6
Bosnia and Herzegovina (BA)	3
Bahamas (BS)	3
Egypt (EG)	1

Table 5.2: Disk images categorized by country of purchase.

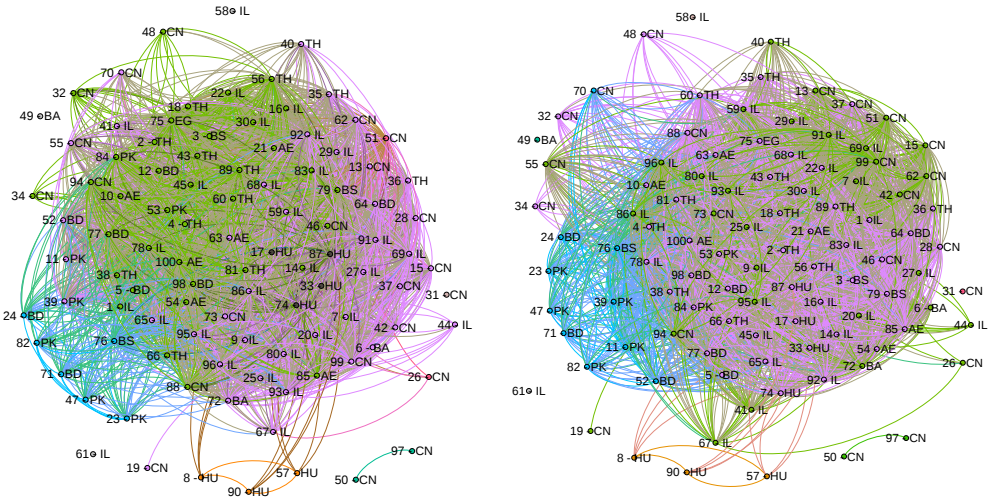
Apple	1
Linux	1
DOS FAT16	12
Win95 FAT32	32
NTFS	24
pCache	1
Unknown	29

Table 5.3: Disk images categorized by partition type.

We still do not know what is causing the underlying similarities between images or which similarity score is better representing the dataset. In an attempt to identify a correlation between the images and metadata associated with each image, we collected the country of origin (Table 5.2) and partition type (Table 5.3) for each image. The majority of the images were purchased in Israel (IL), China (CN), and Thailand (TH) with 30, 22, and 13 images originating from each country, respectively. Visualizing the dataset with the metadata overlay is helpful in identifying patterns within clusters by examining different categories.

To determine if one metric better represents the high-level features of our dataset, we created four graphs (one for each metric for both country of origin and partition type) to see if one better aligned the overlay with the structure and grouping of each graph. We first created two graphs for both the multiset and set Jaccard similarity scores showing the country of purchase for each image (Figure 5.3a and Figure 5.3a). However, given the large tightly connected component of the graph and the diversity in the country of purchase for each image, categorizing the images based on country provided little correlation except in the cases of exact matches (see Section 5.2.2) and interesting cases as in the two isolated nodes labeled CN in the bottom right-hand corner of Figure 5.3 (see Section 5.2.4).

Second, we constructed two graphs for both the multiset (Figure 5.4a) and set (Figure 5.4b) Jaccard similarity scores showing partition type as determined by the `mmls` command. In these graphs, we begin to see correlations emerge between the colored communities and partition type. For example, within the largest cluster, images created with NTFS partitions tend to cluster around other NTFS partitions. We also see Win95 FAT32 and DOS FAT16 partitioned images placed toward the outside of the large cluster and into the outer edges of



(a) Multiset Jaccard Similarity (b) Set Jaccard Similarity

Figure 5.3: Side-by-side comparison of multiset and set Jaccard similarity graphs. These graphs represent each node as the image's number and country of origin with edge weights as calculated using the multiset (left) and set (right) Jaccard similarity metrics. The graph was positioned through the Fruchterman Reingold algorithm [16] and color-coded based on the modularity class algorithm of Blondel [17]. Notice the similarity in overall structure between the two graphs, despite the fact that the edge weights were calculated differently. In contrast, the community coloring of the left graph appears to distinguish partition types more effectively.

the graph. Since a fairly high number of partitions are unknown, it is difficult to know how well the clustering is identifying communities based on partition type, but we do begin to notice patterns emerge from some underlying characteristic of the images (e.g., partition type, operating system, applications).

The apparent link between partition type and the communities identified by the modularity class algorithm from Blondel *et al.* ultimately persuaded us to favor the set similarity metric over the multiset similarity scores. Figure 5.4 shows a side-by-side comparison of the multiset and set Jaccard similarity graphs with the partition-type overlay. We can see the

structure of the two graphs are quite similar. Both contain a large cluster of similar images in the center of the graph and the placement of the images on the outer edge are very similar. However, differences appear in how the community detection algorithm determined the classifications (i.e., coloring) between our similarity metrics. The algorithm detected 10 communities for our multiset calculation and eight communities for our set calculation. With 29 unknown partition types, its difficult to tell how many communities actually exist within our dataset. However, in general, the algorithm correctly grouped the NTFS and DOS FAT16 partition types in their own community but split the Win95 FAT32 partition types across the NTFS and DOS FAT16 communities. This could mean images with Win95 FAT32 partition types are more similar (and have overlapping sector-level hashes) with the NTFS and DOS FAT16 partition types making it difficult for the algorithm to properly classify a community for Win95 FAT32 partitions.

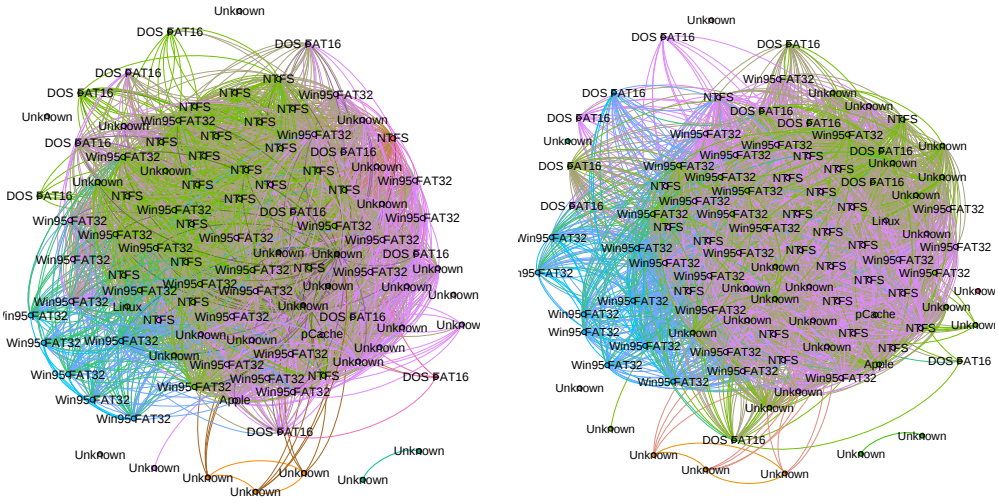
5.2 Experiment 2: Common Set of Hashes Across Dataset

Our dataset did not contain any hashes that are present on every image, as shown in Figure 5.3, where some images are not connected to any other in our dataset (similarity score equal to zero for all other images). However, we could see a large portion of the images were highly connected and clustered together towards the middle of the graph and appeared to be fully-connected. Our second experiment attempted to identify a common set of sectors shared by the disk images in this connected cluster, as well as to examine the meaning of other distinctive structures in the similarity graph.

5.2.1 Fully-Connected Component

Using the k-core filter in Gephi, we were able to determine the largest fully-connected component within our dataset—a subgraph containing 67 images. We hypothesized that there was at least one sector-level hash contained on all images in the fully-connected subgraph given all non-zero similarity scores. However, after completing the intersection of each image in the subgraph, there were not any hashes that were common to all images in this subgraph. This was surprising: we expected to find that at least one sector would be shared across the entire set due to similar operating system files or low-entropy artifacts.

Because the fully-connected component of our graph based on similarity scores did not



(a) Multiset Jaccard Similarity (b) Set Jaccard Similarity

Figure 5.4: Side-by-side comparison of multiset and set Jaccard similarity graphs. These graphs represent each node as the image's partition type with edge weights as calculated using the multiset (left) and set (right) Jaccard similarity metrics. The graph was positioned through the Fruchterman Reingold algorithm [16] and color-coded based on the modularity class algorithm of Blondel [17]. Notice the similarity in overall structure between the two graphs, despite the fact that the edge weights were calculated differently. In contrast, the community coloring of the left graph appears to distinguish partition types more effectively.

contain sector hashes present on each image, we believed low weight edges may affecting our perception of the graph. Given that an edge is drawn between images even if only one hash matches, it is entirely possible that low weight edges may be adding lots of edges. As we filter each graph showing edge weights only above the mean (shown in Figure 5.5), we start to see which images are more strongly connected. If only a small number of hashes match between images, the graph draws an edge between the two images. However, by removing the low edge weights results, we start to see which images are more similar drawing our attention again to the large cluster in the middle and the images which are exact matches around the edges.

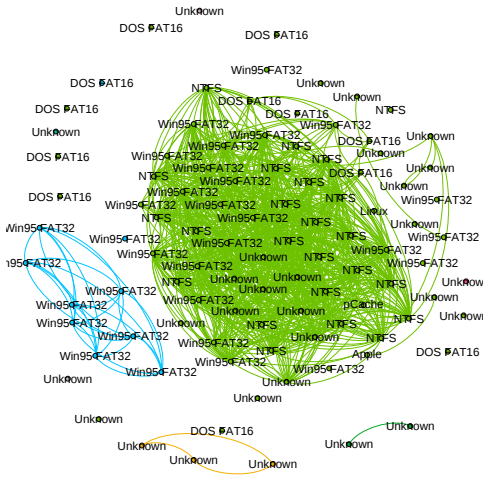


Figure 5.5: Set Jaccard similarity visualization by partition type with edge weight filter. This graph represents each numbered node by the country of its purchase, showing only edge weights above the mean (0.0053746) as calculated using the set Jaccard similarity. The graph was positioned through the Fruchterman Reingold algorithm [16] and color-coded based on the modularity class algorithm of Blondel [17].

5.2.2 Exact Matches

Investigating our dataset, we found three sets of exactly matching images (two sets of two images and one set of three images). The first set of two images included image number 23 and 82 (see the bottom left of Figure 5.4, where these two nodes are connected by a thick cyan edge). The intersection of these two hash databases contained 30 total hashes with 12 distinct hashes. These two images were both purchased in Pakistan (PK) and had Win95 FAT 32 partition types. Interestingly, the second set of exactly matching images, numbered 24 and 71, also contained 30 total hashes with 12 distinct hashes. However, these two images were both purchased in Bangladesh (BD) but did also have Win95 FAT 32 partition types. Each image was 10 gigabytes in size, signifying most of the sectors were zero-filled and were skipped by the bulk_extractor scanner, which ignores empty sectors by default. These four images, in addition to image 47 (bought in PK with a Win95 FAT 32 partition type) are closely grouped on the outer edge of Figure 5.4, and loosely grouped with the highly connected component of the graph. Further examination in Autopsy showed each drive was most likely formatted (very few sector hashes) and had four separate partition

tables that were invalid or unrecognized.

The third set of matching images contained three images numbered 8, 57, and 90. The intersection of these three images contained 8,847,360 total hashes with only 6 distinct hashes. These three images were purchased in Hungary (HU) and had unknown partition types. While the filesystem was corrupted, it appeared these drives were all formatted and had repeating sectors. Further examination of the hexdump of each image showed the 10-character string "#ERASED# " repeated over all non-zero sectors of the disk with the string "by CHG" added at the end of certain sectors. It appears each disk was formatted using the same technique of writing the string continuously throughout the disk. The 5 different alignments of the "#ERASED# " string combined with the final signature produced the six distinct sector types we observed.

5.2.3 No Matches

Four images were not matches to any drive in the dataset including image number 31 (CN), 49 (Bosnia and Herzegovina (BA)), 58 (IL), and 61 (IL). All had unknown partition types, corrupted filesystems, and no discernible strings output except image 61. It appeared to have Windows XP installed based on its header information, but was either formatted or corrupted based on our attempts to examine its file system.

5.2.4 Interesting Matches

Two additional images, 50 and 97, were matches only to themselves and their similarity scores were quite high (multiset similarity of 0.75 and set similarity of 0.60). Interestingly, these two images were not connected to any other images in the entire dataset, were both purchased from China (CN), and both had an unknown partition type. Each had a corrupted filesystem and no discernible strings (most likely due to `strings` not supporting unicode for Chinese characters).

5.3 Experiment 3: Frequency and Distribution of Common Hashes

Up to this point, we have seen that there does not exist a hash which is present on every image in the dataset, nor within the fully-connected subgraph, but we have seen a large number

of similar images within the large cluster and edge cases by using a low-weight edge filter to identify highly similar connections. To identify which high-level similarities are being represented by the matches, we begin by examining the frequency and distribution of our commonly-occurring sectors. Figure 5.6 shows the cumulative distribution function (CDF) of hashes that appeared on more than one disk image and the number of images that contained each hash. The total number of hashes that appear on more than one disk image in our original set of 100 images is 99,412,503, with the 2 most frequently occurring hash appearing on 63 drives. This confirms our observation that there is no single hash present on every image, either in the original 100 or in the highly connected component of 67 nodes. Further, the majority of duplicates are produced by hashes that are themselves relatively rare. Almost 50% (46,843,448) of hashes appear on only two drives, and 90% of hashes appear on eight or fewer drives across the whole set. Hashes that appear on more than one drive are few.

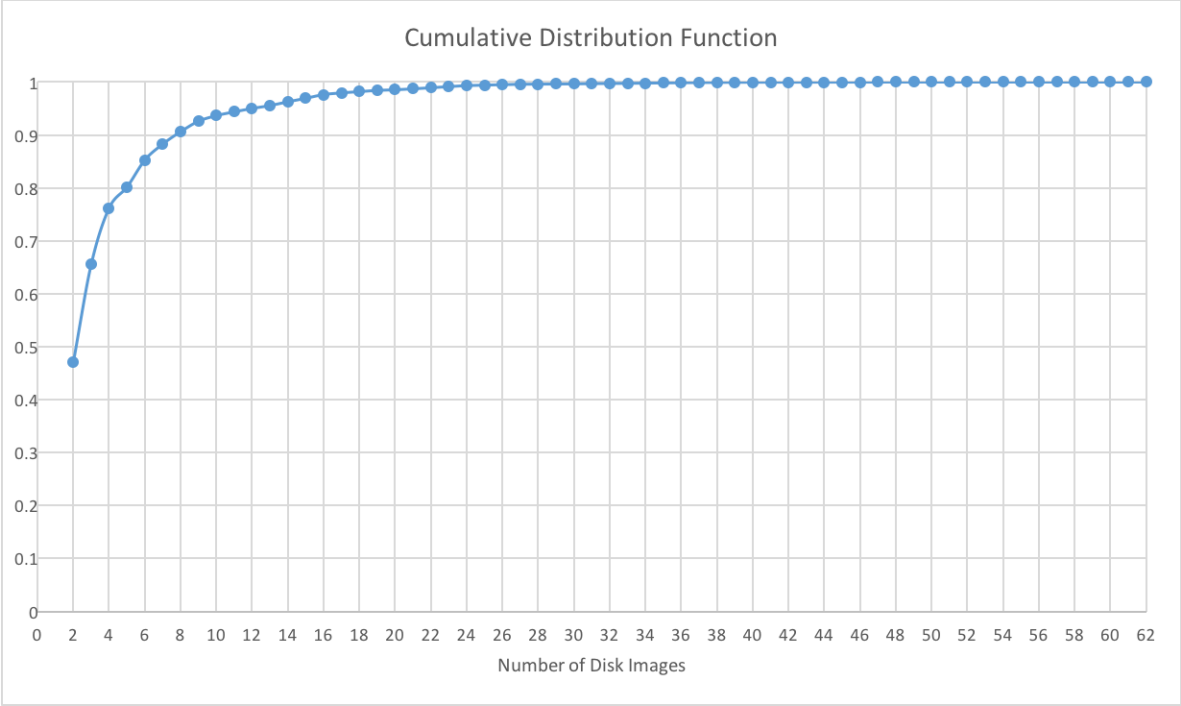


Figure 5.6: This graph shows the cumulative distribution function of duplicate sector hashes by the number of disk images. The largest number of duplicate sector hashes starts with two images and reaches 98.5% at around 20 images.

5.4 Experiment 4: Matching Sector-Level Hashes Across Dataset

With the observation that there were no sector hashes that appeared on every image of our fully-connected subgraph, and no major subset of sectors was responsible for a majority of the edges, we inspected the sector hashes represented in the thin tail of Figure 5.6, corresponding to the sectors that appear on the greatest number of drives. While no one hash contained in every image of the fully-connected subgraph, hashes frequently occurring should tend to be very common files within the dataset. Given that the majority of the partition types are Windows-related, we hypothesized that large counts of sector hash matches occurred as a result of common system files installed on each image.

We scanned individual images against our known set of duplicate hashes from Experiment 3 in SectorScope to identify images containing frequently occurring contiguous sector matches. Scanning image seven against our database resulted in a high number of contiguous sector matches on 56 other images. The large number of sector matches among this grouping of 57 images allowed us to determine which files correlate to these matching sectors and show up frequently among a large grouping of images. The histogram output for our scanned image (shown in Figure 5.7) is at the highest zoom level and represents the entire disk image. Most of the sector hash matches appear at the beginning of the disk image and indicate the location of system files. The height of the histogram is a function of the number of matching hashes in a single bin (i.e., a pixel width, which represents a fraction, depending on the zoom level, of the total disk image length) and the number of times each match appears in the database. At the greatest magnification, where one bin represents a single sector, the height of the bar indicates how many disk images in the dataset contained the hash at that location (since our hash database was designed to contain no more than one copy for each disk image.

We started by investigating the highest number of matching contiguous sectors of our scanned image seven against our database of known duplicate hashes focusing on the 56 images we knew to be highly similar. Close inspection of the hex dump of these frequently occurring matching sector hashes of the scanned image revealed that every sector of the highest matching hashes started its hex dump with hex "4D 5A," which is the magic number for MS-DOS, OS/2, and MS Windows (magic numbers are the first bits of a file which

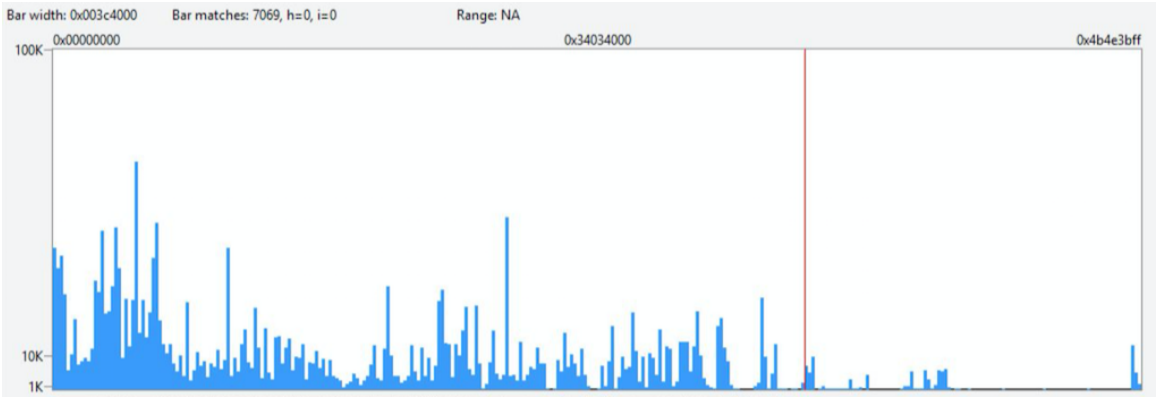


Figure 5.7: SectorScope histogram for scanned image against database of known duplicate hashes. SectorScope shows the number of sectors matching as a histogram moving through the sectors of the hard disk image along the x-axis. At the highest zoom level, the histogram represents the entire disk image. The height of the histogram is a function of the number of matching hashes in a single bin (i.e., a pixel width, which represents a fraction, depending on the zoom level, of the total disk image length) and the number of times each match appears in the database. Most of the sector hash matches appear at the beginning of the disk image and indicate the location of system files.

uniquely identify the type of file). While our current toolset does not provide a trivial way to move between hashes of sectors and files, we were able to locate and determine the files associated with the highest matching sector hashes by identifying the offset in SectorScope and correlating it with the filesystem output of Autopsy. The first 10 files associated with the highest matching sector hashes were common DLL files for the Windows operating system. These 10 files appeared on at least 52 of the 57 images being analyzed. This confirmed our hypothesis that common system files for the same operating system would have the same hashes, which increases the similarity score between images of similar operating systems. The 57 identified images had partition types of DOS FAT16 (6), Win95 FAT32 (20), NTFS (14), and Unknown (17), which confirms the likelihood of seeing common system files based on Windows-related operating systems.

Interestingly, as we started to examine the duplicate hashes with lower match-rates, it became apparent that versions of common windows files were distinguishable. For example, examining the version of command.com led to only 32 images sharing the same default

operating system shell version 6.22. Also of interest, some of the more common windows executables (e.g., format.exe, scandisk.exe, find.exe, fdisk.exe, msbackup.exe) started to match sector-level hashes of other images purchased from the same country. In our SectorScope scans of images purchased in IL, we started to see these executables' sector hashes match only with other images from the same country, which could possibly explain differences in common system files due to language differences between operating system versions.

5.5 Experiment 5: Common Set of Hashes by Community

We identified the operating system of our 57 images from Experiment 4 (see Section 5.4) to see if sector-level hashes were distinguishable between each grouping. In doing so, the summary of operating systems included Windows 95 (2), Windows 98 (12), Windows XP (26), and Unknown (17), as shown in Table 5.4.

Windows 95	2
Windows 98	12
Windows XP	26
Unknown	17

Table 5.4: Operating system counts of our subset of 57 images.

	Multiset	Set
Windows 95	258,027	46,360
Windows 98	551,126	189
Windows XP	251,021	172
Unknown	238,254	14

Table 5.5: Count of hashes in the intersection of all drives in our subset of 57 images with common operating systems.

We then graphed the set Jaccard similarity scores with the partition and operating system using the Force Atlas [44] layout algorithm to determine if operating system was a better indicator than partition type as an overlay (see Figure 5.8 for the overall structure and community coloring). Upon examining the layout, we noticed the images containing Windows XP were tightly clustered in the left-center and Windows 95 and Windows 98

	Windows 95	Windows 98	Windows XP	Unknown
Windows 95	258,027	156,246	208,690	233,315
Windows 98		551,126	379,240	376,709
Windows XP			251,021	402,632
Unknown				238,254

Table 5.6: Count of hashes in the multiset intersection of all drives in our subset of 57 images when grouped by operating systems. Note that the number of hashes in the intersection declines significantly, indicating that drives with the same operating system have more in common with each other than they do with drives of other operating systems.

	Windows 95	Windows 98	Windows XP	Unknown
Windows 95	46,360	75	69	10
Windows 98		189	56	5
Windows XP			172	11
Unknown				14

Table 5.7: Count of distinct hashes in the set intersection of all drives in our subset of 57 images when grouped by operating systems. Here as in Figure 5.6, we see a decline in common hashes in the intersections of disk images with different operating systems, as compared to the intersections of images in the same system. Working with types instead of tokens highlights this relationship even more clearly.

were clustering around the edges, which confirmed our belief that operating systems files were contributing to similarity scores (see Figure 5.9 for the magnified portion of the graph). While the partition type contributes to the overall grouping of the graph, searching by operating system yields a better explanation for the concentration of clustering.

We then analyzed the intersecting hashes of each operating system as multisets and sets (Table 5.5). The higher counts in the multiset column when compared to the set column indicates each grouping contains a large number of duplicate hashes. The Windows 95 grouping has significantly higher amounts of locally distinct hashes (46,360) than the other groupings; the small number of locally distinct hashes within the unknown operating systems (14) most likely indicates a diverse set of operating systems included in the group. We calculated the intersection between each operating system to determine similarity between

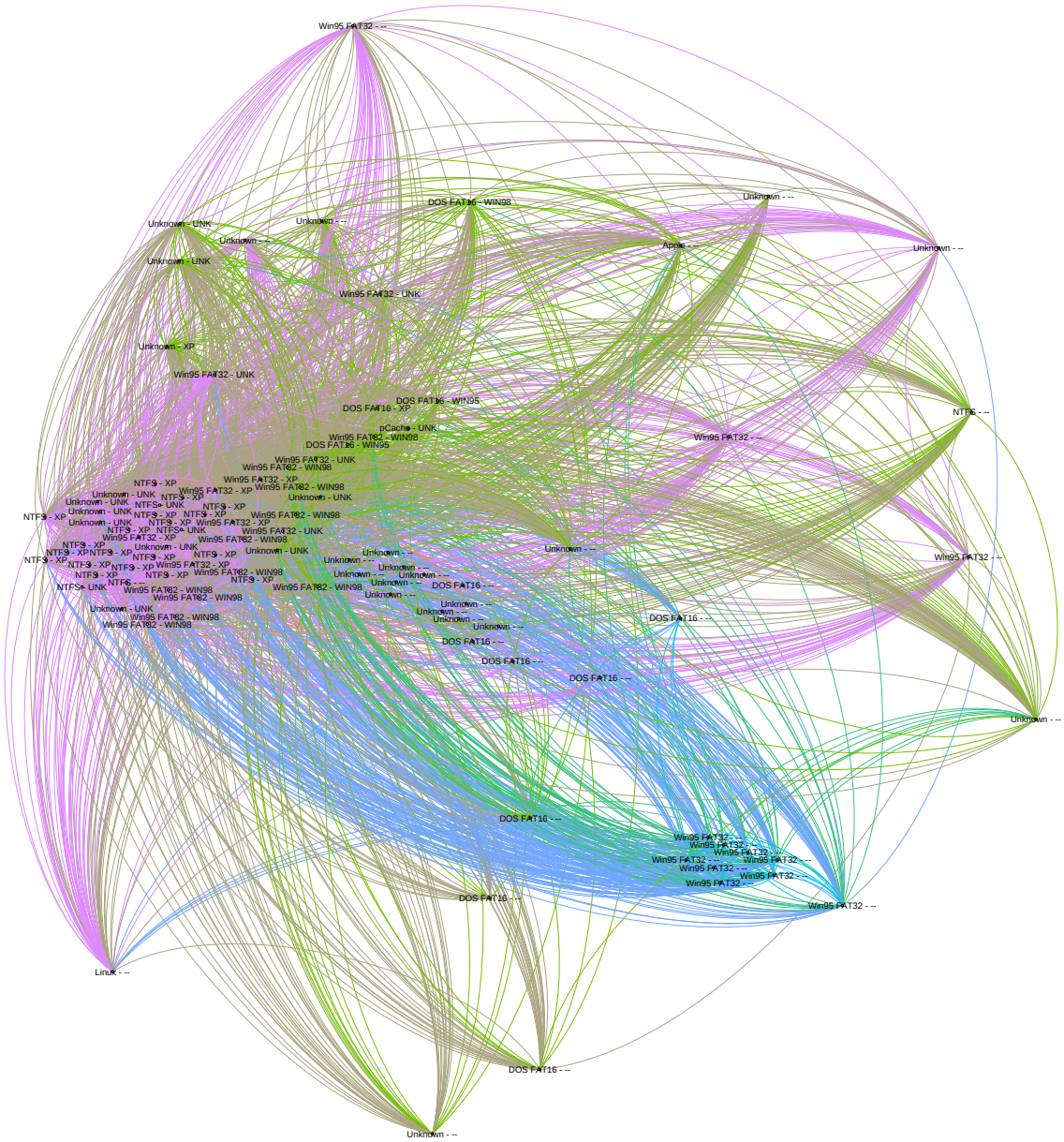


Figure 5.8: Set Jaccard similarity visualization by partition type and operating system. This graph is provided to show the overall structure and coloring. The graph was positioned through the Force Atlas algorithm [44] and color-coded based on the modularity class algorithm of Blondel [17].

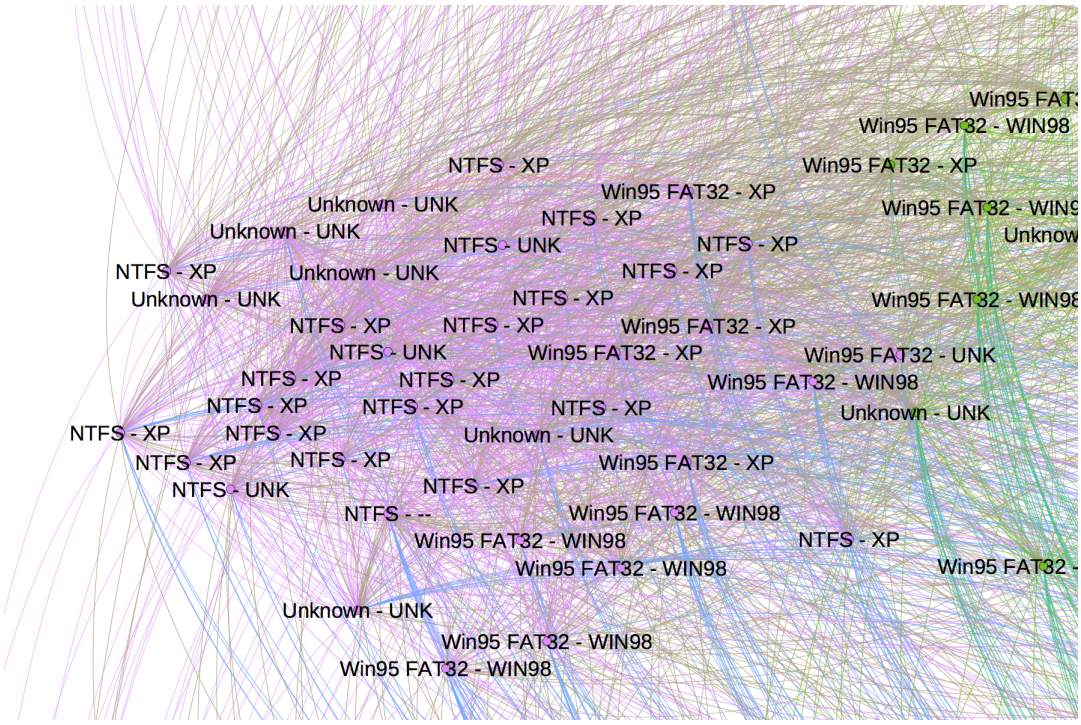


Figure 5.9: Set Jaccard similarity visualization of partition type and Windows operating system. This graph is a zoomed in portion of Figure 5.8. It shows clustering of images containing Windows XP images in the center and Windows 95 and Windows 98 moving outward to the right. The graph was positioned through the Force Atlas algorithm [44] and color-coded based on the modularity class algorithm of Blondel [17].

communities in Table 5.6 (allowing duplicate hashes) and Table 5.7 (showing only distinct hashes). We again see a large number of duplicate hashes between each grouping of operating system. However, we can now identify matching hashes exclusive to each operating system and matching hashes that overlap between operating systems. The number of hashes in the intersection declines significantly, indicating that images with the same operating system have more in common with each other than they do with images of other operating system versions. The only multiset intersection that does not decrease is between Windows XP and the unknown operating systems. In addition, the set intersection between Windows XP and the unknown operating systems includes 11 of the 14 locally distinct hashes contained in the unknown operating system. We believe it is possible that a high number of the unknown operating systems actually contain Windows XP and our inability to determine

their operating systems is causing us to separate the two communities. This provides a possible method of identifying an unknown image by intersecting individual images against our sets of known communities and examining the resulting number of distinct hashes.

CHAPTER 6:
Conclusion

6.1 Summary

Digital forensic practitioners need the ability to compare mass amounts of digital media in a practical and efficient manner. As the amount of digital media grows, so to do the domains in which digital forensics can be applied. Law enforcement, intelligence agencies, and incident response teams in military and business organizations need a way to correlate large amounts of digital media to determine similarity, detect patterns, and effectively triage large volumes of digital evidence.

Traditional digital forensic methods involved individual analysts examining individual hard drive images. The accelerated growth rate of digital media and increase in cyber incidents has forced practitioners to move toward automated tools and bulk analysis. This shift in approach has evolved from using cryptographic hashes to determine matches of entire files, to segments of files, to blocks of raw data. The introduction of hash-based carving by Garfinkel and McCarrin [24] suggests the possibility of developing methods that avoid the need to parse the filesystem for exact matches of logical files. Since the reliance of matches has moved beyond the individual file, or pieces of the file, matches are based off blocks of data and are filesystem agnostic.

In our experiment, we built on the concepts employed in prior research into hash-based carving to create hash databases representing a subset of 100 hard disks in the RDC. Using set theory operations, we calculated pair-wise comparisons of each drive in the subset to create 4,950 similarity scores for a multiset and set Jaccard similarity metric ranging between zero and one. We graphed the results representing images as nodes and edge weights as the similarity score and determined the similarity score was an effective means to explore our dataset.

The graph provided key insights into our dataset, showing a large majority of the images were highly connected among each other (of our 100 images, 67 were fully connected), a small number of the images had no connections, and a small number of the images

were tightly connected with only each other. However, there was no single hash that was represented in every image. Therefore, we examined the most frequently seen hashes to determine which sectors were matching between the images. Scanning individual images against a database of known duplicate hashes, we were able to identify a subset of 57 images that contained high numbers of contiguous matching sector-level hashes. Further examination of these 57 images showed the top 10 most occurring sector hashes represented were common DLL files from the Microsoft Windows operating system. Each of these 10 common system files was represented on at least 52 of the 57 images, and provided partial explanation as to the underlying cause of similarity. Further examination of the subgraph allowed the identification and grouping by operating system. In doing so, we were able to identify hashes exclusive to certain versions of operating system as well as sector hashes that matched across different versions of operating system. Examination of these hashes leads to potential possibilities of categorizing newly acquired disk images by operating system and version by establishing known communities of matching hashes.

We find that sector-based matching is sufficient to identify drives sharing common DLLs, which indicates similarity in their operating systems, and could therefore be used to corroborate deductions regarding characteristics of information systems associated with the target media. This capability may provide insight into the community infrastructure, behavior or characteristics of organizations from which drives were collected. For example, matching a newly acquired drive to a cluster of similar drives might give a quick indicator of how recently an organization has updated or patched their software. If these operating system-related similarities were to be filtered out, this approach might prove useful in identifying other important similarities across large collections of drives, such as virus infections or common user data, though this experiment was beyond the scope of the current work (in part because there is no such relationship expected among drives in the RDC).

6.2 Future Work

There are many areas of further investigation suggested by our analysis. The sample size was restricted by computational limits associated with our tools. Our analysis would, otherwise, benefit from a more diverse set of images. One recommendation is to use our methodology with known images representing a larger population, especially with respect to operating systems and versions. Based on our findings, sector hashes of known operating

system and version should exhibit higher similarity scores. Matching against clusters where the base operating system is known could lead to better categorization of images. In our initial experiments, we tested against used drives to simulate the environment likely to be encountered by forensic analysts. However, in future work researchers hoping to discover operating system signatures might instead create a dataset of images containing unused operating system installations with various patch levels to create more informative clusters, and perhaps ultimately to build signatures of sectors that characterize each operating system version.

In addition, while our methodology faces limitations in processing pair-wise comparisons for large amounts of disk images, it would be beneficial to expand our methodology into a cloud computing infrastructure where the set operations is conducted in the cloud with a local hash database functioning as a cache containing the most important or immediately useful sectors. While hashdb is an adequate tool to store the computed hash databases, computing set intersections, and more importantly set unions, can become too large for the software to handle at around 1.5 billion hashes. The identification of a set of hashes that could be used as a noise filter is one example of a case where the process of identifying the hashes might require distributed computation, but the outcome might be stored in a local hashdb instance.

The calculation of similarity metrics and visualization suggest a wealth of possibilities around exploratory analysis. Our analysis incorporated only three categories of metadata (country of origin, partition type, and operating system), and only began to scratch the surface on possible techniques to discern differences in common operating system software. Similar visualization strategies may have applications for prototyping automated analytical methods, as well as presenting results to an analyst in an intuitive manner.

Lastly, manual inspection of correlating known sector hashes found in SectorScope with its filesystem in Autopsy is incredibly time intensive and does not scale. Automated techniques capable of identifying files associated with matching sector-level hashes would greatly enhance the ability to find similarities of interest between images.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX: [Disk Image Filenames by Number]

1	IL007-0014
2	TH03-0003
3	BS001-0016
4	TH03-0055
5	BD1-1044
6	BA0-1132
7	IL3-0220
8	HU001-0018
9	IL28
10	AE10-1038
11	PK1-1021
12	BD1-1006
13	CN26-10
14	IL3-0129
15	CN27-31
16	IL004-0013
17	HU001-0016
18	TH03-0041
19	CN33-65
20	IL2-0082
21	AE1001-1007
22	IL3-0172
23	PK1-1032
24	BD1-1062
25	IL005-0004
26	CN6-20
27	IL19
28	CN9-07
29	IL3-0169

30	IL005-0006
31	CN26-20
32	CN18-37
33	HU001-0010
34	CN27-42
35	TH0001-0054
36	TH03-0004
37	CN16-27
38	TH03-0014
39	PK1-1079
40	TH0001-0064
41	IL3-0096
42	CN34-16
43	TH03-0068
44	IL2-0003
45	IL007-0008
46	CN28-04
47	PK1-1050
48	CN27-37
49	BA0-1117
50	CN32-46
51	CN8-01
52	BD1-1010
53	PK1-1081
54	AE1001-1008
55	CN11-22
56	TH0001-0009
57	HU001-0002
58	IL3-0148
59	IL3-0154
60	TH03-0019

61	IL3-0209
62	CN32-57
63	AE10-007
64	BD1-1019
65	IL4-0007
66	TH0001-0108
67	IL2-0073
68	IL3-0113
69	IL3-0164
70	CN19-07
71	BD1-1069
72	BA0-1131
73	CN3-04
74	HU001-0021
75	EG1001-0001
76	BS001-0004
77	BD1-1011
78	IL3-0181
79	BS001-0024
80	IL3-0206
81	TH0001-0011
82	PK2-2003
83	IL006-0005
84	PK1-1015
85	AE1001-1001
86	IL2-0090
87	HU001-0019
88	CN16-33
89	TH0001-0043
90	HU001-0009
91	IL2-0084

92	IL54
93	IL2-0049
94	CN30-25
95	IL21
96	IL13
97	CN32-72
98	BD1-1007
99	CN33-97
100	AE10-1020

Table A.1: Disk image name represented by number

List of References

[1] T. Wilson. (2012, July). Study: Cybersecurity market to double in next five years. [Online]. Available: http://www.darkreading.com/study-cybersecurity-market-to-double-in-next-five-years/d/d-id/1137973. Accessed February 8, 2016.

[2] National Institute of Standards and Technology. (2012, July). Digital evidence. [Online]. Available: http://www.nist.gov/oles/forensics/digital_evidence.cfm. Accessed July 21, 2015.

[3] Computer Forensics. (2013, February). [Online]. Available: https://www.us-cert.gov/security-publications/computer-forensics. Accessed February 8, 2016.

[4] National Institute of Justice. (2010, November). Digital evidence and forensics. [Online]. Available: http://www.nij.gov/topics/forensics/evidence/digital/Pages/welcome.aspx. Accessed July 23, 2015.

[5] M. Brown, D. Stikvoort, K.-P. Kossakowski, G. Killcrece, R. Ruefle, and M. Zajicek. (2003, April). Handbook for computer security incident response teams (csirts). [Online]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=6305. Accessed January 4, 2016.

[6] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton. (2014, April). The digital universe of opportunities: Rich data and the increasing value of the internet of things. [Online]. Available: http://www.emc.com/leadership/digital-universe/2014iview/index.htm. Accessed July 14, 2015.

[7] E. Grochowski and R. D. Halem, "Technological impact of magnetic hard disk drives on storage systems," *IBM Systems Journal*, vol. 42, no. 2, p. 338, 2003. [Online]. Available: http://search.proquest.com/docview/222428908?accountid=12702

[8] S. Haldar and A. Aravind, *Operating Systems*. Panchsheel Park, New Delhi: Pearson India, May 2010.

[9] Seagate. Transition to advanced format 4k sector hard drives. [Online]. Available: http://www.seagate.com/tech-insights/advanced-format-4k-sector-hard-drives-master-ti/. Accessed July 14, 2015.

[10] M. Russinovich, D. A. Solomon, and A. Ionescu, *Windows Internals Part 2*. Redmond, Washington: Microsoft Press, 2012.

[11] Forensics Wiki. (2008, July). Disk image. [Online]. Available: http://forensicswiki.org/wiki/Disk_image. Accessed February 25, 2016.

[12] ASR Data's Expert Witness Compression Format. (2002). [Online]. Available: http://forensicswiki.org/wiki/ASR_Data's_Expert_Witness_Compression_Format. Accessed March 8, 2016.

[13] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. IEEE In Compression and Complexity of Sequences (SEQUENCES '97)*, 1997, pp. 21–29.

[14] L. Wetzel. (2014). Types and tokens. [Online]. Available: http://plato.stanford.edu/archives/spr2014/entries/types-tokens/. Accessed February 25, 2016.

[15] A. Rajaraman, J. Leskovec, and J. D. Ullman, *Mining of Massive Datasets*. Cambridge, United Kingdom: Cambridge University Press, 2014.

[16] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Software: Practice and Experience*, vol. 21, pp. 1129–1164, 2006.

[17] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[18] A. Z. Broder, "Some applications of rabin's fingerprinting method," in *Sequences II: Methods in Communications, Security, and Computer Science*, 1993, pp. 143–152.

[19] R. Rivest. (1992, April). The md5 message-digest algorithm. [Online]. Available: http://tools.ietf.org/html/rfc1321. Accessed July 14, 2015.

[20] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *Fast Software Encyrption*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 371–388.

[21] X. Wang and H. Yu, *How to Break MD5 and Other Hash Functions*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 19–35.

[22] National Institute of Standards and Technology Special Publication 800-168. Approximate Matching: Definition and Terminology. (2014, May). Computer Security Resource Center (CSRC). [Online]. Available: http://csrc.nist.gov/. Accessed July 21, 2015.

[23] S. Garfinkel, A. Nelson, D. White, and V. Roussev, "Using purpose-built functions and block hashes to enable small block and sub-file forensics," *Digital Investigation*, vol. 7, pp. S13–S23, 2010.

[24] S. Garfinkel and M. McCarrin, "Hash-based carving: Searching media for complete files and file fragments with sector hashing and hashdb," *DFRWS*, 2015.

[25] P. Jaccard, "The distribution of the flora in the alpine zone," *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912. [Online]. Available: http://dx.doi.org/10.1111/j.1469-8137.1912.tb05611.x

[26] R. Merkle. (1979, June). Secrecy, authentication, and public key systems. [Online]. Available: http://www.merkle.com/papers/Thesis1979.pdf. Accessed February 24, 2016.

[27] M. Rabin, "Fingerprinting by random polynomials," Center for Research in Computing Technology, Harvard University. TR-CSE-03-01, 1981.

[28] R. Vassil, "Data fingerprinting with similarity digests," in *Advances in Digital Forensics VI*, ser. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2010, vol. 337, pp. 207–226. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15506-2_15

[29] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *SRC Technical Note*, 1997.

[30] Federal Evidence Review. (2002). [Online]. Available: http://federalevidence.com/blog/2008/september/using-‘‘hash’’-values-handling-electronic-evidencet. Accessed March 8, 2016.

[31] National Institute of Standards and Technology. National software reference library. [Online]. Available: http://www.nsrl.nist.gov/

[32] N. Harbour. (2002). Defense computer forensics lab. [Online]. Available: http://dcfldd.sourceforge.net/. Accessed July 14, 2015.

[33] L. Chen and G. Wang, "An efficient piecewise hashing method for computer forensics," *International Workshop on Knowledge Discovery and Data Mining*, pp. 635–638, 2008.

[34] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital Investigation*, vol. 3, pp. 91–97, 2006.

[35] Jesse Kornblum. (2015). ssdeep. [Online]. Available: http://ssdeep.sourceforge.net/. Accessed March 8, 2016.

[36] V. Roussev. (2013). [Online]. Available: http://roussev.net/sdhash/sdhash.html. Accessed March 8, 2016.

[37] S. Garfinkel, "Digital media triage with bulk data analysis and bulk_extractor," *Computers & Security*, vol. 32, pp. 56–72, February 2013.

[38] S. Collange, Y. S. Dandass, M. Daumas, and D. Defour, "Using graphics processors for parallelizing hash-based data carving," in *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, Jan 2009, pp. 1–10.

[39] S. Garfinkel, "DFRWS 2006 challenge report," 2006. [Online]. Available: http://sandbox.dfrws.org/2006/garfinkel/part1.txt. Accessed August 12, 2015.

[40] J. Young, K. Foster, S. Garfinkel, and K. Fairbanks, "Distinct sector hashes for target file detection," *IEEE Computer*, vol. 45, pp. 28–35, 2012.

[41] S. Garfinkel, P. Farrella, V. Roussev, and G. Dinolta, "Bringing science to digital forensics with standardized forensic corpora," *Digital Investigation*, vol. 6, pp. S2–S11, September 2009.

[42] Forensics Wiki. (2010, June). Bulk extractor. [Online]. Available: http://www.forensicswiki.org/wiki/Bulk_extractor. Accessed August 5, 2015.

[43] B. Allen. (2015). Nps hash database library. [Online]. Available: https://github.com/simsong/hashdb. Accessed August 5, 2015.

[44] Gephi. (2015). Download. [Online]. Available: https://gephi.org/users/download/. Accessed January 11, 2016.

[45] B. Allen. (2015). Sectorscope. [Online]. Available: https://github.com/NPS-DEEP/NPS-SectorScope. Accessed August 5, 2015.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California